Coroutines

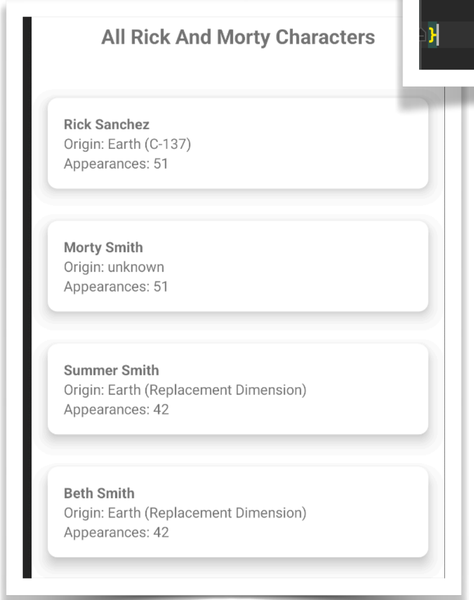
לפעמים אנו מבצעים פעולות כבדות אשר יכולות לתקוע את הMain Thread (התהליך הראשי עליו האפליקציה רצה) ולגרום לאפליקציה לא להגיב, על מנת למנוע זאת, את הפעולות הכבדות אנחנו מוציאים לפועל בתהליך מקביל לMain Thread

נתבונן בדוגמא:
קיבלנו API של ריק ומורטי, עלינו לבצע בקשה מהשרת על מנת לקבל מערך של דמויות מהסדרה

```
class MyViewModel : ViewModel() {

    private val repository: Repository = Repository()
    private val _rickAndMortyLiveData: MutableLiveData<RickAndMortyCharacters> = MutableLiveData()
    val rickAndMortyLiveData: LiveData<RickAndMortyCharacters> = _rickAndMortyLiveData

    init {
        viewModelScope.launch { this: CoroutineScope
            val characters = repository.getCharacters()
            _rickAndMortyLiveData.postValue(characters)
        }
    }

}
```

```
interface RickAndMortyDao {
    @GET("character")
    suspend fun getCharacters(): RickAndMortyCharacters
}
```

הוספת suspend לפעולה
הופכת את הפעולה לCoroutine




```
class Repository {

    suspend fun getCharacters(): RickAndMortyCharacters {
        return ApiManager.rickAndMortyDao.getCharacters()
    }
}
```

כעת, בViewModel נשתמש בפעולה בתוך Coroutine Scope
שימוש בפעולות Coroutine אפשרי רק בתחום של Coroutine

`androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.0`

ספרייה המכילה תחומים שונים של Coroutine (ביניהם viewModelScope)

MainScope

תחום Coroutine שניתן לשימוש בכל מקום,
פעולות ברירת מחדל יוצאות לפועל על התהליך הראשי!

```
MainScope().launch {    this: CoroutineS

}
```

withContext

פעולה המקבלת הקשר של Coroutine
ונותנת תחום בו ניתן להשתמש בפעולות Coroutine

```
return withContext(viewModelScope.coroutineContext) {    this:
    val chars = repository.getCharacters()
    chars    ^withContext
}
```

Dispatchers

ניתן לעבור בין תהליכים בתחום Coroutine
על ידי שימוש ב Dispatchers

```
MainScope().launch {    this: CoroutineScope

    // IO Thread - IO Operations
    withContext(Dispatchers.IO) {    this: CoroutineScope

    }
    // Main Thread - UI Operations
    withContext(Dispatchers.Main) {    this: CoroutineScope

    }
}
```

async

פעולה הנותנת אופציה לחכות לCoroutine עד לקבלת תשובה
באמצעות ()await

```
// suspend function which awaits result
suspend fun getCharacters(): RickAndMortyCharacters {
    return async {    this: CoroutineScope
        return@async viewModel.getCharactersAwait()
    }.await()
}

// use the function to reach results after received
getCharacters().results
```