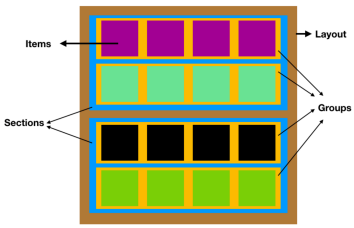
# CheetSheet – UICollectionView

## Compositional Layout




Hierarchie Compositional Layout:
Item>group>section>layout

4 Klassen, die implementiert werden müssen, um ein Compositional Layout zu bauen:

- `NSCollectionLayoutSize` —width & height dimensions sind vom Typ `NSCollectionLayoutDimension` und kann defnierit warden, indem die fractional width/height des Layouts gesetzt wird (Prozent ist relative zum Container), oder eine absolute/geschätzte Zeit gesetzt wird
- `NSCollectionLayoutItem` — Layout der Zelle
- `NSCollectionLayoutGroup` — hat das `NSCollectionLayoutItem` in einer horizontalen, verticalen oder benutzerdefinierten Form
- `NSCollectionLayoutSection` — Initialisierung der Section, indem die `NSCollectionLayoutGroup` übergeben wird

Beispiel zur Implementierung der Klassen in einer Collection View:

```
collectionView = UICollectionView(frame: view.bounds,
collectionViewLayout: createLayout())
```

```
private func createLayout() -> UICollectionViewLayout {

//1
let itemSize = NSCollectionLayoutSize(widthDimension:
.fractionalWidth(1.0),heightDimension: .fractionalHeight(1.0))

let item = NSCollectionLayoutItem(layoutSize: itemSize)

//2
let groupSize = NSCollectionLayoutSize(widthDimension:
.fractionalWidth(1.0),heightDimension: .absolute(44))

//3
let group = NSCollectionLayoutGroup.horizontal(layoutSize:
groupSize,subitems: [item])

let section = NSCollectionLayoutSection(group: group)

let layout = UICollectionViewCompositionalLayout(section: section)

return layout

}
```

Spacing hinzufügen:

```
item.contentSize = NSDirectionalEdgeInsets(top: 5, leading: 5,
trailing: 5, bottom: 5)

group.interItemSpacing = .fixed(CGFloat(10))
```

Multiple Layouts → layoutEnvironment:

```
func createLayoutDiffSection() -> UICollectionViewLayout {
        let layout = UICollectionViewCompositionalLayout { (sectionIndex: Int,
            layoutEnvironment: NSCollectionLayoutEnvironment) -> NSCollectionLayoutSection?

            var columns = 1
            switch sectionIndex{
            case 1:
                columns = 3
            case 2:
                columns = 5
            default:
                columns = 1

            }
```

## Diffable Data Source

DataSource mit UICollectionDiffableDataSource erstellen:

1. Snapshot erstellen: UICollectionViewDataSource<Section, Int>
2. Snapshot mit Sections und Items befüllen, die geupdated werden sollen: appendSections() /appendItems()
3. Snapshot anwenden: apply()

Beispiel:

```
func configureDataSource(){
    dataSource = UICollectionViewDiffableDataSource<Section, Int>(collectionView:
        self.collectionView){
        (collectionView, IndexPath, number) -> UICollectionViewCell? in

        guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
            NumberCell.reuseIdentifier, for: IndexPath) as? NumberCell else {
            fatalError("Cannot create new cell")
        }
        cell.label.text = number.description

        return cell
    }

    var initinalSnapshot = NSDiffableDataSourceSnapshot<Section, Int>()
    initinalSnapshot.appendSections([.main])
    //Create an array of Snapshots that serve as our data
    initinalSnapshot.appendItems(Array(1...100))

    dataSource.apply(initinalSnapshot, animatingDifferences: false)

}
```

Eventuell: Hashable Protocol implementieren, damit DiffableDataSource über die UUID() die Items identifizieren kann, die von einem Update Cycle zum nächsten gleich sind.

```
private let identifier: UUID
func hash(into hasher: inout Hasher) {
    hasher.combine(self.identifier)
}
```