



BoostCamp iOS

MOGAY

Contents

— — —

Auto Layout 소개

SignUp View 예시

Advanced Auto Layout

Auto Layout?

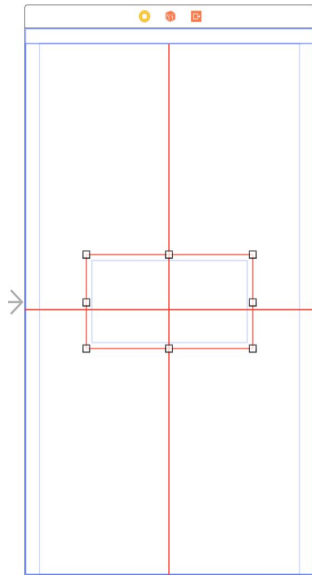
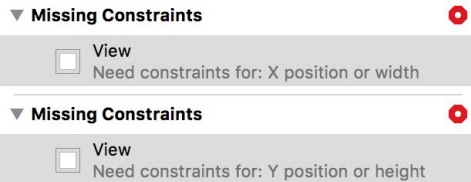
View에 배치 된 'Constraint'에 따라 View 계층 구조의 모든 View의 크기와 위치를 동적으로 계산

Intrinsic Content Size

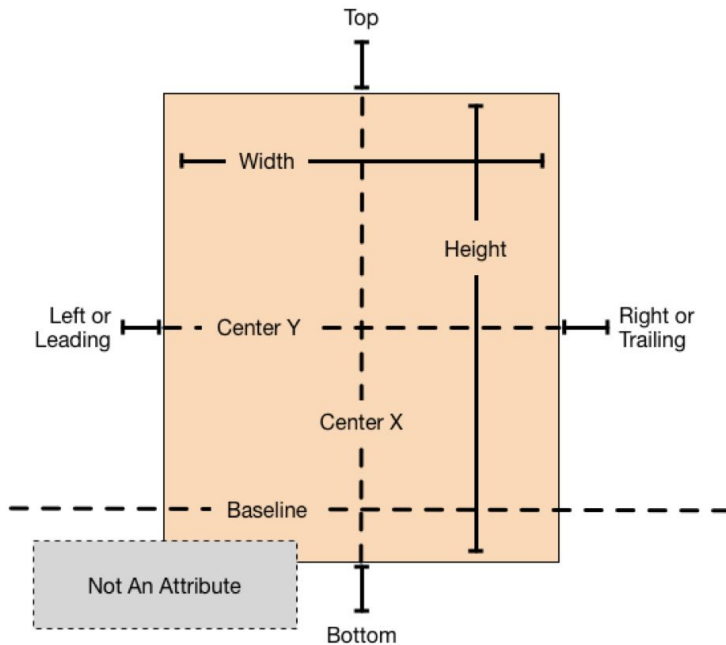
“뷰의 핵심적인 내용을 모두 표시할수 있는 가장 작은 영역의 크기”
대부분의 뷰는 콘텐츠 고유 사이즈를 갖는다

컨테이너 역할을 하는 UIView넣고

오토레이아웃을 중앙으로 잡으면 오류!!



Auto Layout Attributes



leading, trailing <-> left, right
차이는 무엇일까?

leading은 일반적으로 **left**,
trailing은 일반적으로 **right**를 의미하지만,
오른쪽에서 왼쪽으로 쓰는 언어로 설정되어있는
경우에는 **leading**이 **right**를 의미한다. 따라서
다국어를 지원하는 어플리케이션을 개발할 경우
주의해서 사용해야 한다.

Constraint



$$\underbrace{\text{RedView}}_{\text{Item 1}}.\underbrace{\text{Leading}}_{\text{Attribute 1}} = \underbrace{1.0}_{\text{Multiplier}} \times \underbrace{\text{BlueView}}_{\text{Item 2}}.\underbrace{\text{trailing}}_{\text{Attribute 2}} + \underbrace{8.0}_{\text{Constant}}$$

- **NSLayoutConstraint Class**

`NSLayoutConstraint(item: redView, attribute: .leading, relatedBy: .equal, toItem: blueView, attribute: .trailing, multiplier: 1.0, constant: 8.0)`

- **Layout Anchors**

`redView.leadingAnchor.constraint(equalTo : blueView.trailingAnchor, constant : 8.0)`

- **Visual Format Language**

```
let views = ["blueView":blueView, "redView":redView]
```

```
let formatString = "[blueView]-8-[redView]-]"
```

```
let constraints = NSLayoutConstraint.constraints(withVisualFormat: formatString, options: [], metrics: nil, views: views)
```

Priority

— — —

Constraint 의 우선순위

UIPriority에서 설정가능 (1~1000)

Required(1000), High(750), medium(500) ,Low(250)의 기본값이 있음

여러가지 우선순위로 제약조건이 걸리면 가장 큰 값의 제약조건이 활성화 된다.

Content Hugging & Compression Resistance

- Content Hugging

content does not want to grow

Content Hugging은 고유 사이즈의 **최대 크기**에 제한을 두는 것

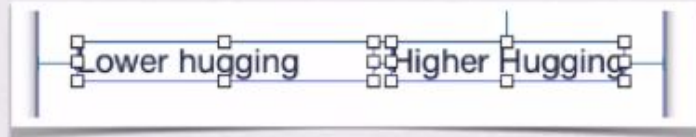
- Compression Resistance

content does not want to shrink

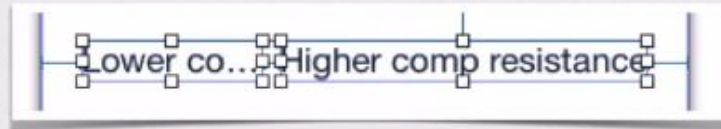
Compression Resistance는 **최소 크기**에 제한을 두는 것

Hugging vs. Resistance

✦ **Content Hugging:** Don't want to grow



✦ **Content Compression Resistance:** Don't want to shrink



raywenderlich.com

<https://www.raywenderlich.com/64392/video-tutorial-beginning-auto-layout>

Auto Layout in SignUp View (Multiplier)

View 중에 하나를 기준으로 정함 (Image View)

기준으로 정한 것의 위치를 '확실하게' 정해 줌 Top Layout Guide

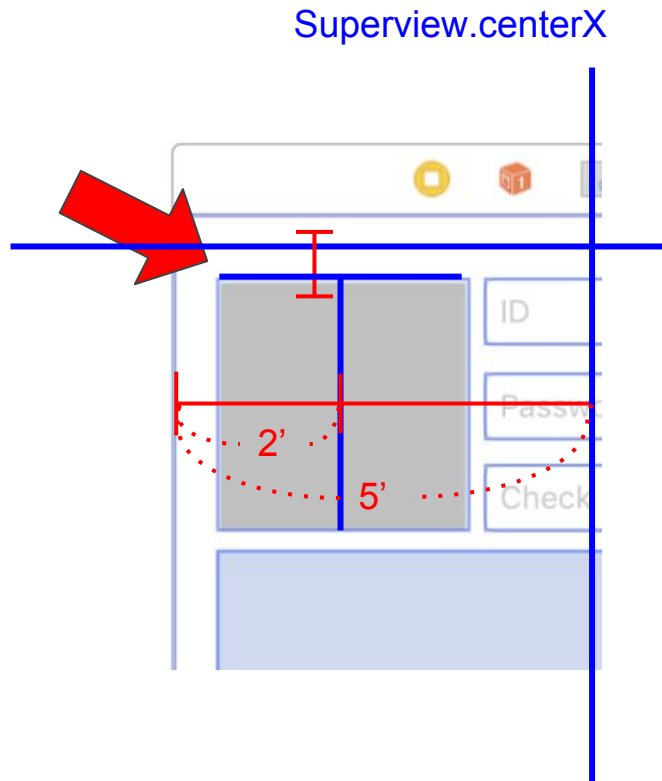
top, centerX, width, width-height aspect

Image View.top = Top Layout Guide.bottom + 8

Image View.centerX = 0.4 x Superview.centerX

Image View.width = 0.3 x Superview.width

Image View.width = Image View.height



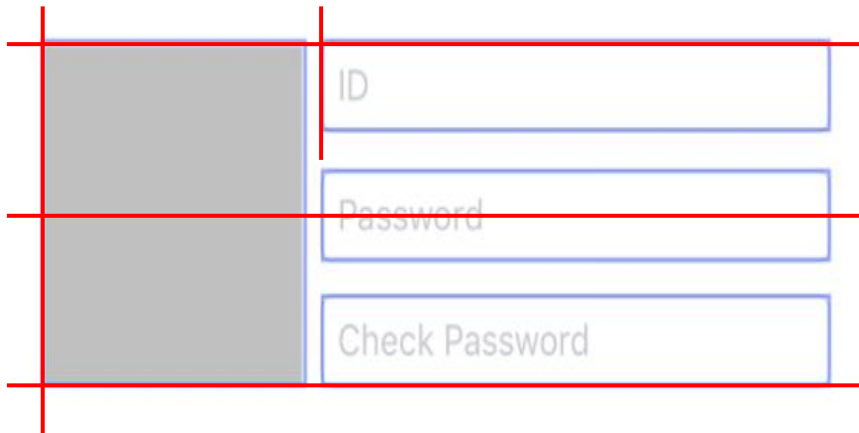
Auto Layout in SignUp View (Multiplier)

Image View를 기준으로 이웃 TextField
Constraint 추가

`Id Field.top = Image View.top`

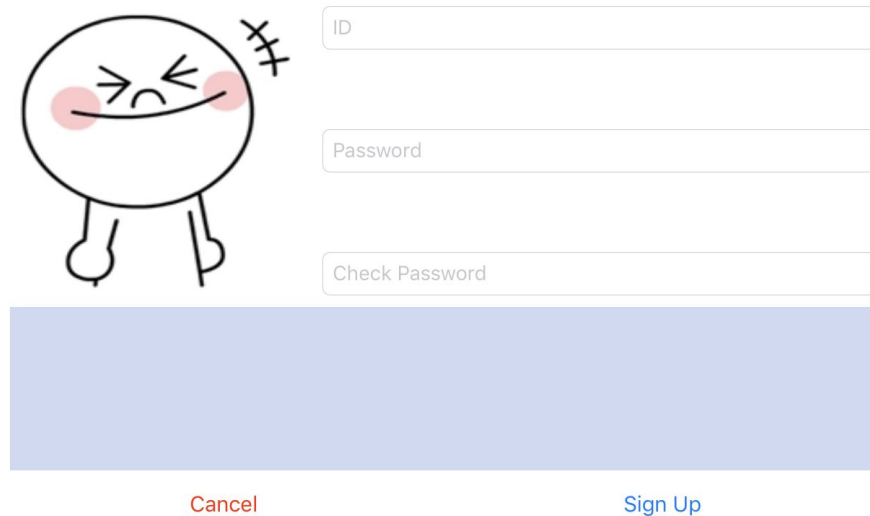
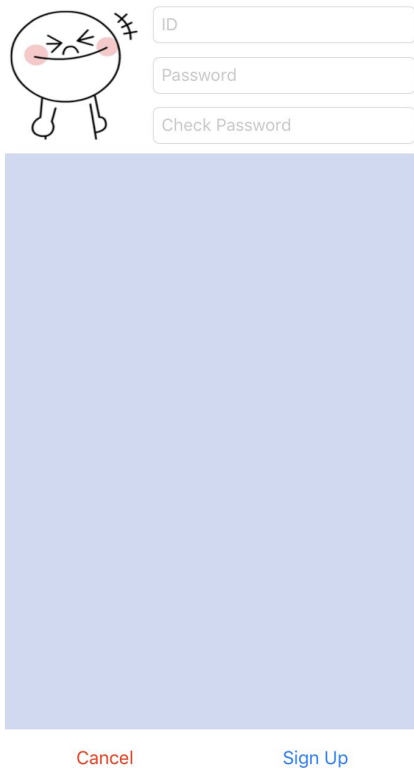
`Id Field.width = 0.575 x Superview.width`

`Id Field.leading = 7.5 x Image View.leading`



Auto Layout in SignUp View (Multiplier)

— — —



Auto Layout in SignUp View (Stack View)

stack view 에서

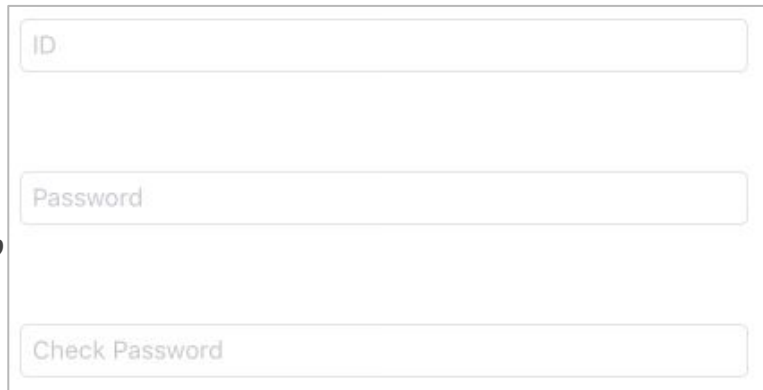
textfield의

default height를 유지하고 싶다면,

distribution 프로퍼티에

equal spacing 혹은

equal centering 값을 준다.



The diagram shows a vertical stack of three text input fields. The top field is labeled 'ID', the middle field is labeled 'Password', and the bottom field is labeled 'Check Password'. Each field is a rounded rectangle with a light gray border and a light gray background. The fields are stacked vertically with equal spacing between them, demonstrating the 'equal spacing' distribution property.

Auto Layout in SignUp View (Stack View)

Stack View의 Distribution에 대해서...

➤ fill

➤ fillEqually

➤ fillProportionally

➤ equalSpacing

➤ equalCentering

Distribution in Stack View

fill

스택 뷰 < 정렬된 뷰

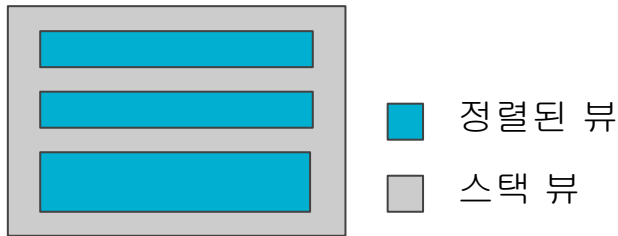
➡ resistance priority 에 따라 줄어든다.

정렬된 뷰 < 스택 뷰

➡ hugging priority 에 따라 늘어난다.

둘다 모호할 때

➡ arrangedSubview 배열의 인덱스 기반으로 재 정렬



Distribution in Stack View

— — —

fillEqually

같은 간격으로 정렬

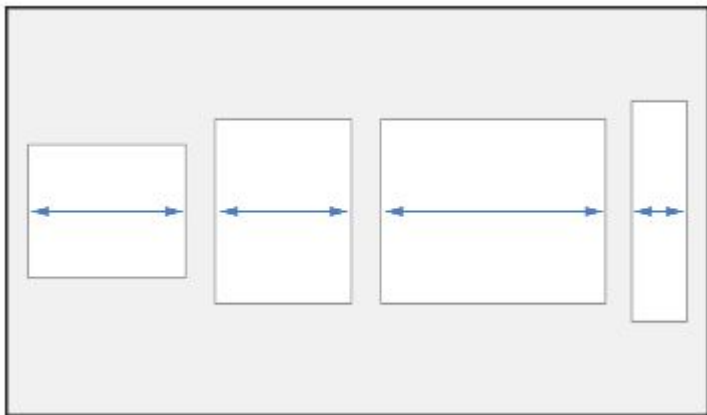
fillProportionally

intrinsic content size에 비례하여 재 정렬

Fill- Group

— — —

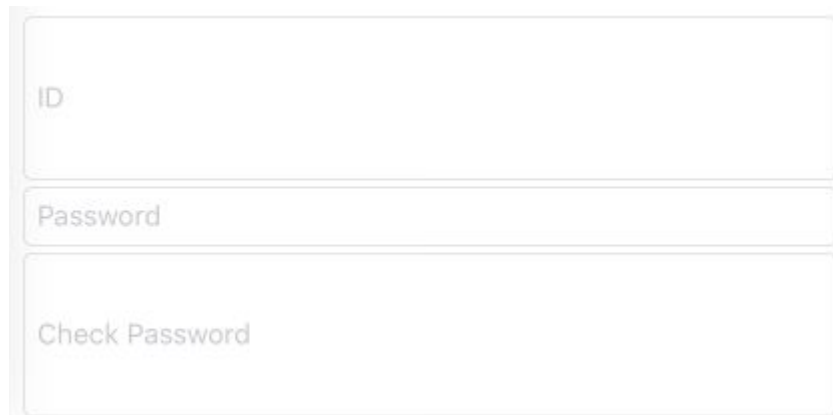
뷰를 늘리고 줄인다.



fill versus fillEqually

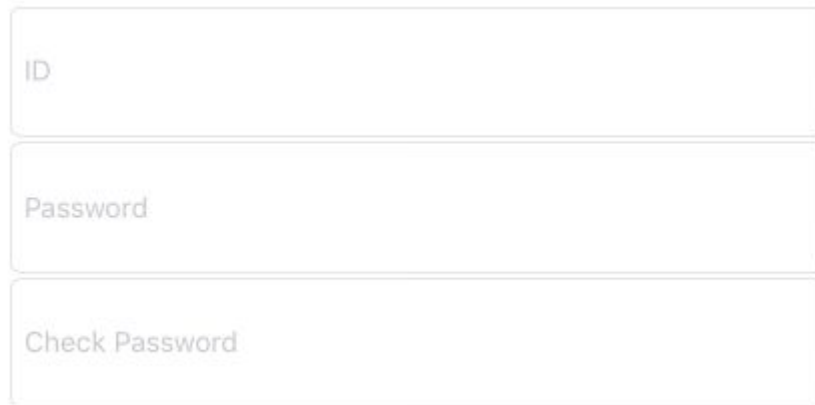
— — —

spacing -> 3



A form layout created using the `fill` method. It consists of three stacked rectangular input fields. The top field is labeled "ID", the middle field is labeled "Password", and the bottom field is labeled "Check Password". The fields are separated by a vertical spacing of 3 units.

fill



A form layout created using the `fillEqually` method. It consists of three stacked rectangular input fields. The top field is labeled "ID", the middle field is labeled "Password", and the bottom field is labeled "Check Password". The fields are separated by a vertical spacing of 3 units.

fillEqually

Distribution in Stack View

equalSpacing

정렬된 뷰 < 스택 뷰 : 동등하게 간격을 준다.

스택 뷰 < 정렬된 뷰 : `resistance priority`에 따라 줄인다.

둘 다 모호할 때 : `arrangeSubviews` 배열의 인덱스에 기반하여 재 정렬

equalCentering

뷰 들의 `vertical` 축 사이에 동등한 간격을 준다.

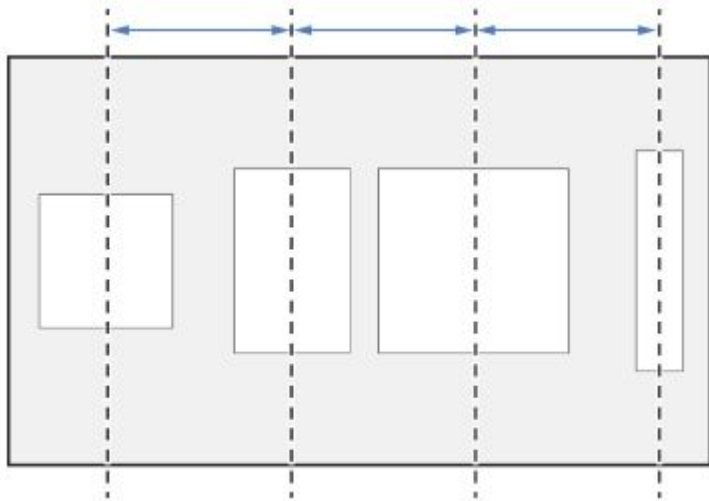
이렇게 정렬 했는데 스택 뷰에 맞지 않다면,

1. 최소 `sapcing` 프로퍼티 간격으로 줄인다.
2. `resistance priority`에 따라 줄인다.
3. `arrangeSubviews` 배열의 인덱스에 기반하여 재 정렬

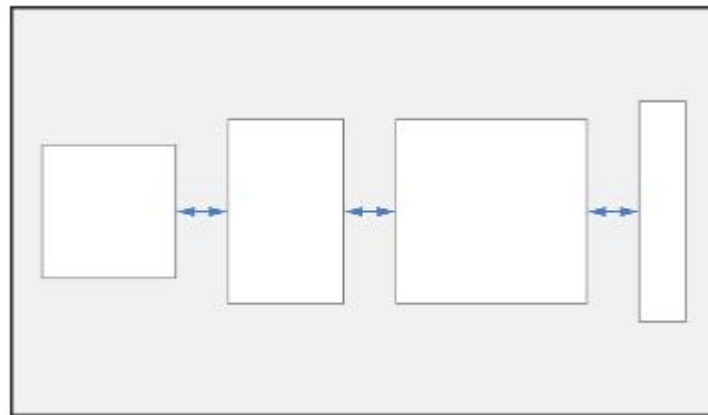
Equal- Group

— — —

case equalCentering = 4



case equalSpacing = 3



TableViewCell 의 높이 변경하기

- 기본적으로

table view cell의 height는 table view delegate's `tableView:heightForRowAtIndexPath:method` 에 의해 결정된다.

- table view cell 의 높이를 바꾸고 싶으면

1. table view의 `rowHeight`라는 property를 `UITableViewAutomaticDimension`로 설정.
2. table view의 `estimatedRowHeight`라는 property에 값을 설정.

```
tableView.estimatedRowHeight = 85.0
```

```
tableView.rowHeight = UITableViewAutomaticDimension
```

: 이러한 프로퍼티 값을 설정하면, 시스템이 **table view cell**의 높이를 계산하는데 **Auto Layout**을 사용하게 된다.

Cumulative

Constraints는 누적되는 성질을 지님.

— — —

- 추가하면 누적

: 너비 제약을 가지고 있는 뷰에 새로운 너비 제약을 추가해도 이전에 존재하던 너비 제약이 업데이트/삭제 안됨.

- 서로꼬이면 오류발생

: 하나의 뷰에 동일한 속성에 대한 제약이 2개 이상 존재하고, 각 제약의 우선 순위가 동일하다면 레이아웃 오류 발생.

- 확인 후 제거

: 새로운 제약을 추가하기 전에 동일한 성격과 우선 순위를 가지는 제약이 존재하는지 확인한 다음 직접 제거.

이전 제약을 제거하고 새로운 제약을 추가하는 방식은 오류를 발생시킬 확률이 높고 성능상 불리함.

- 속성 변경을 통해서

: 제약의 **constant** 속성을 변경하여 원하는 효과를 얻을 수 있다면 이 방식을 사용하는 것이 좋음.

- 반드시 같은 종류의 제약을 2개 이상 추가

: 각 제약의 우선 순위에 차이를 두고, 각 우선 순위가 조건에 따라 적절히 업데이트 되도록 신경써야 함.

LayoutSubviews() 는 무엇?

— — —

- 역할

:뷰에 적용된 **LayoutConstraints**로 화면을 구성.

- layoutSubviews() 가 implicit하게 호출

layout의 **constant**값이 변경

layout의 **isActive**변경으로 낮은 **priority**값을 가진 같은 속성 **layout**이 존재할 때

view가 추가 삭제 됐을 때

...

Layout관련 메서드 호출 시점

- layoutSubview 는 언제 불리나?

명시적인 호출: `setNeedsLayout()`,
`layoutIfNeeded()`

`setNeedsLayout`: 다음런루프에 `layoutSubviews()` 호출

`layoutIfNeeded`: 제약조건 변화 등에의해 기존 레이아웃에서 변경될 점이 있다면 불림, 변경이 없는경우 불리지 않음.

암시적인 호출:

뷰에 적용된 제약조건이 변경 될 경우 특별히 `setNeedsLayout`을 명시적으로 호출하지 않아도 다음 런루프에서 자동으로 불림.

- updateConstraints 가 언제 불리나?

명시적인 호출: `setNeedsUpdateConstraints()`

호출된 경우 다음 런루프에 불림

암시적인 호출:

`layout` 진행 도중에 업데이트될 제약조건이 있는 경우 시스템 내부적으로 호출됨

* 레이아웃 변경/제약조건 업데이트는 순차적으로만 일어나는것이 아니고 서로 상호적으로 필요에따라 번갈아가면서 호출되며 시스템 내부적으로 다루지는 부분이 많아 정확한 호출 시점을 잡아내는것이 쉽지 않다.

애플 권고 사항

- layoutSubviews

Apple: You should override this method only if the autosizing and constraint-based behaviors of the subviews do not offer the behavior you want. You can use your implementation to set the frame rectangles of your subviews directly.

=> 제약 기반으로한 Subviews들의 마음에 안들면 Subviews의 frame을 조작하라.

- updateConstraints

Apple: It is almost always cleaner and easier to update a constraint immediately after the affecting change has occurred. For example, if you want to change a constraint in response to a button tap, make that change directly in the button's action method.

You should only override this method when changing constraints in place is too slow, or when a view is producing a number of redundant changes.

=> 제약사항 업데이트는 그 영향을 주는 행위 발생시점에 바로 변경이 깔끔.
만약 성능이나 많은 변화가 필요하다면

Override

* 일반적으로는 뷰를 생성하는 단계에서 Layout을 추가. (Controller내에서 생성단계, CustomView는 initializer내)