COCOA CONCEPTS AND COMPONENTS

COCOA SELECTOR I

- A selector is similar to a function pointer but uses a string that identifies the method to call.
- If you have different instances of different classes with the same method, a selector could be used to always call this method (regardless of the type of the class).
- It seems like Apple is moving away from selectors slowly. However, they are still used a lot. We will therefore look at how to use selectors on an interface.

COCOA SELECTOR II

```
//the class NSTimer calls a method after a certain time interval.
//You can also set "repeats" to true to call the method multiple times.
//The following command will call our method (target: self) "mySelectorCall"
//once (repeats: false) after waiting 0.1 seconds.
NSTimer.scheduledTimerWithTimeInterval(0.1, target: self, selector: "mySelectorCall",
        userInfo: nil, repeats: false)

//this method will be called
func mySelectorCall() {
    print("I am called");
}
```

COCOA SELECTOR III

```
//Selector methods can accept parameters. In our NSTimer example, the method signature
//will accept the timer as parameter. When dealing with selectors, you need to check the
//documentation to get the right signature. Please note that when you want to use a
//parameter, you need to add a ":" to the end of your selector string!
NSTimer.scheduledTimerWithTimeInterval(0.1, target: self, selector: "mySelectorCall:",
        userInfo: "Mr X", repeats: false)

//this method will be called
func mySelectorCall(timer : NSTimer) {
        //you can access the userInfo as property from the timer
        let str = timer.userInfo as! String
    print(str + " called me");
}
```

UISCROLLVIEW

- Used to add scrolling functionality to a page.
- Can be extended to support zooming and panning (Use a UIScrollViewDelegate)

```
//create the UIScrollView and the view to scroll in Storyboard
//create two outlets and the following method in your controller
override func viewDidLayoutSubviews() {
        self.myScrollView.contentSize =
                CGSizeMake(myViewToScroll.frame.size.width,
                        myViewToScroll.frame.size.height);
}
```

COCOA DELEGATE I

- A *delegate* can be used to handle certain aspects of an object in another object.
- Typically, components offer other objects to register themselves as delegates. For certain events, the delegate method is called.
- Delegates can have optional or required methods.

COCOA DELEGATE II

```
//A good delegate example is the use of a UITableView to display data in a table. This
//component expects to read the data from a UITableViewDataSource. We define our own
//view controller to be this delegate
class ViewController: UIViewController, UITableViewDataSource {

    //as you can see, we defined the tableview in our storyboard
    @IBOutlet weak var myTableView: UITableView!  //use a UITableView component

    override func viewDidLoad() {
        super.viewDidLoad()

        myTableView.dataSource = self //set the delegate
    }
```

COCOA DELEGATE III

```
//now we can implement the following two methods, which are defined
    //in the data source delegate and called by the table view
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return 3; //return the number of entries in the table view
    }
    //return the cell for the provided index
    func tableView(tableView: UITableView,
        cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let cell : UITableViewCell = tableView.dequeueReusableCellWithIdentifier("cell")!
        cell.textLabel!.text = String(indexPath.row); //set the index number as a label
        return cell;
    }
}
```

EXERCISE

Create a new project and replace the UIViewController by a UITableViewController. A UITableViewController already has a UITableview component. Implement the delegate methods numberOfRowsInSection and cellForRowAtIndexPath. Use a string array with 3-5 entries and display the values in the table.

UITABLEVIEW

- The UITableView is a powerful component. In order to use it, you must implement the UITableViewDataSource delegate (as seen).
- You can also implement the UITableViewDelegate that provides numerous other methods when using the UITableView.

UITABLEVIEW CELL SELECTION

```
//add yourself as UITableViewDelegate (see above)
//then you can implement this method that will be called whenever
//the user clicks on a cell
func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
        print("The following row was selected: #" + String(indexPath.row))
}
```

AUTOLAYOUT PROGRAMMATICALLY

VISUAL FORMAT LANGUAGE I

- Easy way to define multiple constraints using a simple language.
- Grammar is published
- Cannot be used to define all constraints (center is typically problematic) The following slides show a simple example with three buttons (button1, button2 and button3).

VISUAL FORMAT LANGUAGE II

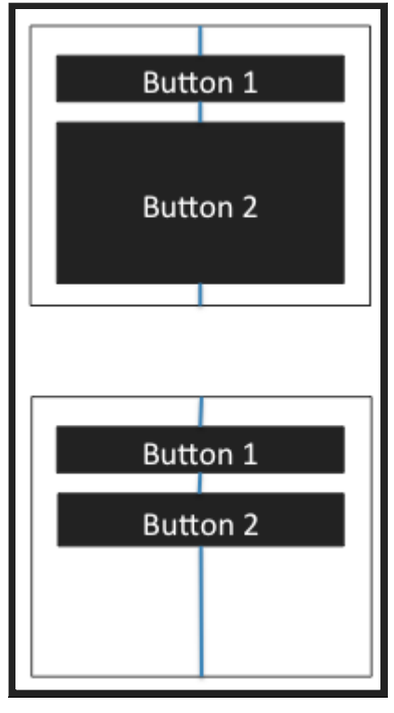
```
//let the two views be on the same row (H: = horizontally)
//with a default distance
H:[button1]-[button2]

//let the two views be on the same column (V: = vertically)
//with a default distance
V:[button1]-[button2]
```

VISUAL FORMAT LANGUAGE III

```
//let button1 start at the left border of the parent view and end at the right border
H:|-[button1]-|
//let button1 start at the top border of the parent view and end at the bottom border
V:|-[button1]-|

//combined
H:|-[button1]-[button2]-|

//let the width or height of button1 be 60 points
[button1(60)]

//let the width or height of button1 be between 60 and 80 points
[button1(>=60,<=80)]

//let the width or height of button1 be the same as button2
[button1(==button2)]
```

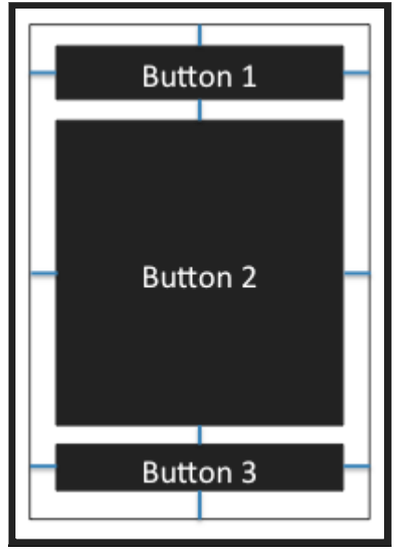
VISUAL FORMAT LANGUAGE IV

```
//Sometimes, you want to have one element that is flexible
//in size. In the following example, button1 will get
//a default height and button2 will be resized according
//to the available space of the view.
V:|-[button1]-[button2(>=20)]-|

//you can of course also be flexible at the distance
//definition. In the following example, the height of
//buttons 1 and 2 will always be default whereas
//additional space is used on the lower border.
V:|-[button1]-[button2]-(>=20)-|
```



VISUAL FORMAT LANGUAGE V

```
//As mentioned before, autolayout constraints need to be
//fully defined. With our three buttons, the following
//would be sufficient
V:|-[button1]-[button2(>=20)]-[button3]-|
H:|-[button1]-|
H:|-[button2]-|
H:|-[button3]-|
```



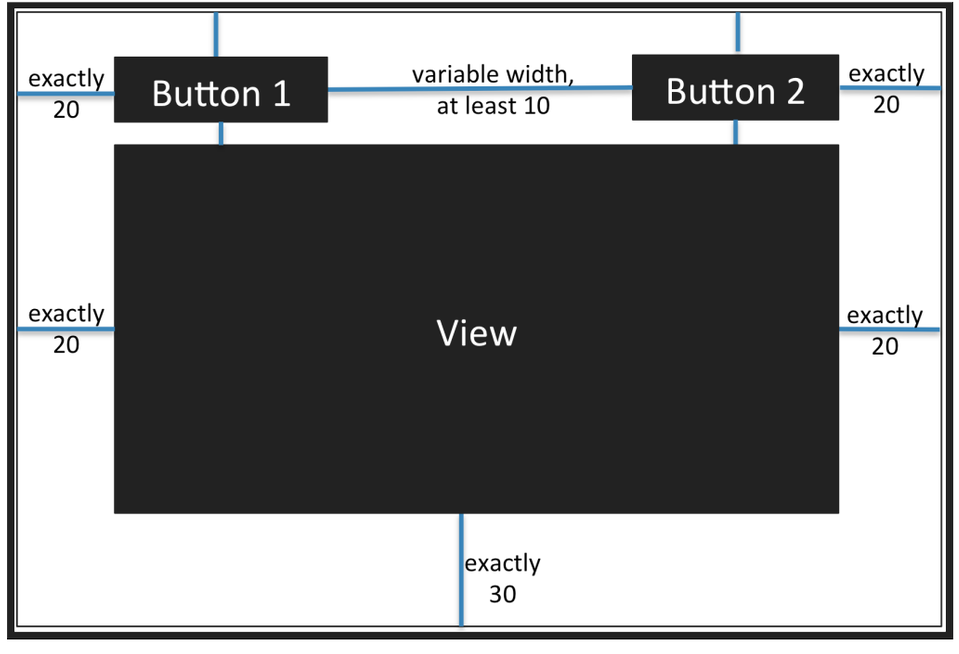
CREATION

```
//create a dictionary with all our views
let viewsDictionary = ["button1": button1, "button2" : button2 ];

//the constraints are always added to the parent container
parentView.addConstraints(
        NSLayoutConstraint.constraintsWithVisualFormat(
                "V:|-20-[button1(100)]-[button2]-50-|",

                //always write 0 when using visual format
                options:NSLayoutFormatOptions(rawValue: 0),

                //can be nil but can be used for constants
                metrics:nil,

                //our directory with all views from the rule
                views:viewsDictionary));
```

METRICS

```
//similar to the viewsDirectory, you can create a metrics
//directory and pass this dictionary to the addConstraints method
let metrics = ["leftMargin": 15.0,
                            "defaultButtonHeight" : 100,
                            "rightMargin" : 50.0 ];

//you can use the defined metrics in your rules
//these constants do not have square brackets (like the views)
"V:|-leftMargin-[button1(defaultButtonHeight)]-[button2]-rightMargin-|",
```

EXERCISE: GUI



Create visual constraints for the GUI.

CGCONTEXT

MOTIVATION OF CGCONTEXT

It is sometimes necessary to have an empty canvas where you can draw whatever you like. iOS provides a CGContext for a UIView that you can use to draw points, lines and curves. (similar to the HTML canvas).

Drawing to the CGContext should always be done in the drawRect method of a UIView.

CGCONTEXT DRAWRECT

```
//Inherit from UIView and override drawRect
class MyDrawingView : UIView {
        override func drawRect(rect: CGRect) {
                //get the context and store it in ctx
                let ctx : CGContextRef = UIGraphicsGetCurrentContext()!;
                //set the stroke color to green
                CGContextSetStrokeColorWithColor(ctx, UIColor.greenColor().CGColor);
                //move to point 10,10 and draw a line to 100,100
                CGContextMoveToPoint(ctx, CGFloat(10), CGFloat(10));
                CGContextAddLineToPoint(ctx, CGFloat(100), CGFloat(100));
                //move to point 100,10 and draw a line to 10,100
                CGContextMoveToPoint(ctx, CGFloat(100), CGFloat(10));
                CGContextAddLineToPoint(ctx, CGFloat(10), CGFloat(100));
                //stroke the path
                CGContextDrawPath(ctx, CGPathDrawingMode.Stroke)
}}
```

EXERCISE: CGCONTEXT

Create a new project and implement a UIView where a filled red circle is drawn. For drawing the circle, use:

```
CGContextFillEllipseInRect(ctx, CGRectMake(x, y, width, height))
```

CGCONTEXT UPDATE

```
//If you want to update the drawn picture, you need to call
//setNeedsDisplay() on the view
func myMethodOnEventX {
        //do something...

    self.setNeedsDisplay()
}

//as you use a normal UIView, all known interaction events are
//available like addGestureRecognizer...
```

EXERCISE: CGCONTEXT UPDATE

Use your project from the last exercise and add an NSTimer with repeat==true. Try to update the circle by either changing the position or size. Hint: You can instantiate the timer in the UIView constructor, when using the storyboard, the constructor