计算器开发实验报告

姓名	学号
张乐简	191220162

概述

本次实验任务为开发一款类似于Mac系统自带计算器的小程序，但仅支持单步运算。我的界面布置基本模仿自带计算器，完成了数字输入、基本运算、科学计算、角弧度切换等功能。接下来，报告将依次介绍实验环境的准备、用户界面搭建、实现计算功能和测试结果。

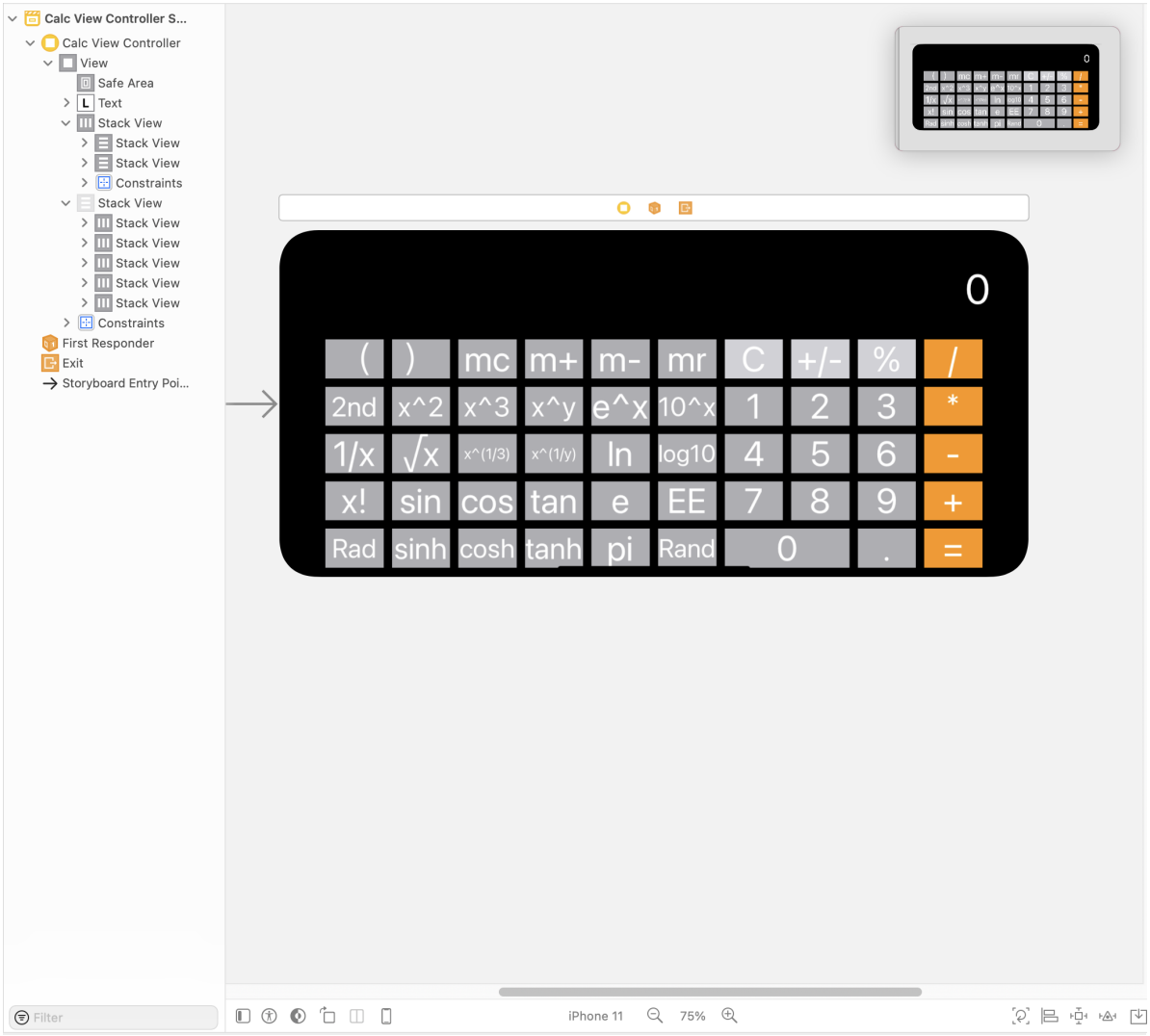
实验过程

环境配置

App的开发一部分在虚拟机上完成，一部分在实验室电脑完成。均采用MacOs Big Sur系统。IDE则选择Xcode。

GUI搭建

按实验要求，依次开发横竖屏两个面板。其中，横屏界面包含了竖屏界面，因此仅介绍横屏界面的搭建以及屏幕切换功能的实现，不单独介绍竖屏界面的搭建。



横屏界面搭建

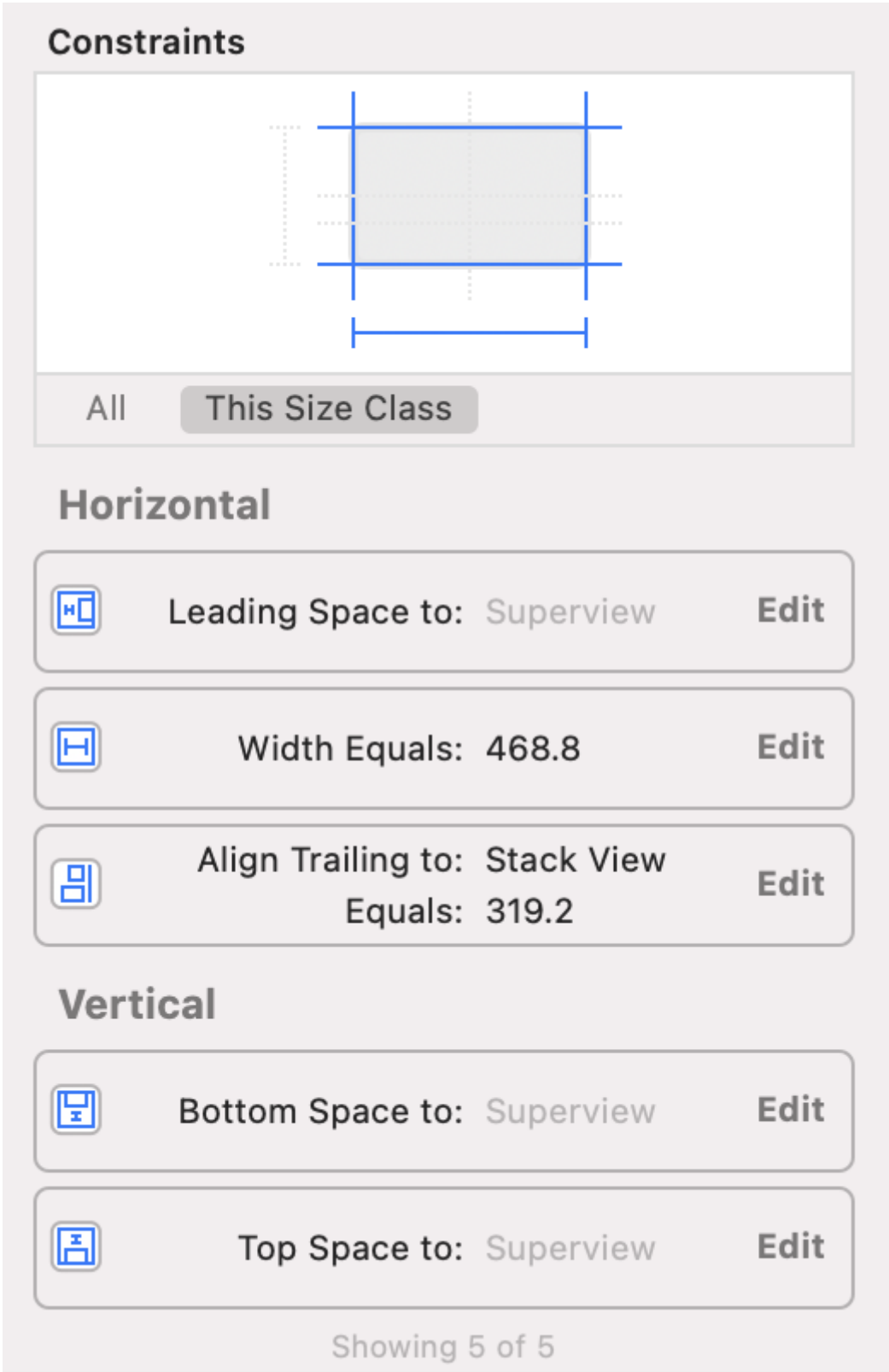
横屏的界面可粗分为三部分：数字显示器、科学计算界面以及数字和基本计算界面。将两个计算界面分开是为了让后者可以直接迁移到竖屏上去。

数字显示器

数字显示器由一个Label组成。为控制相对位置，在横屏模式下给了它一些约束。向上、向左、向右均是与SafeArea保持距离，同时控制宽度在100，这四个约束在横竖屏均不变。另外，还需与下方两个计算界面构成的StackView保持一定的距离。在竖屏时，这变成与数字和基本计算界面保持一定的距离。

科学计算界面

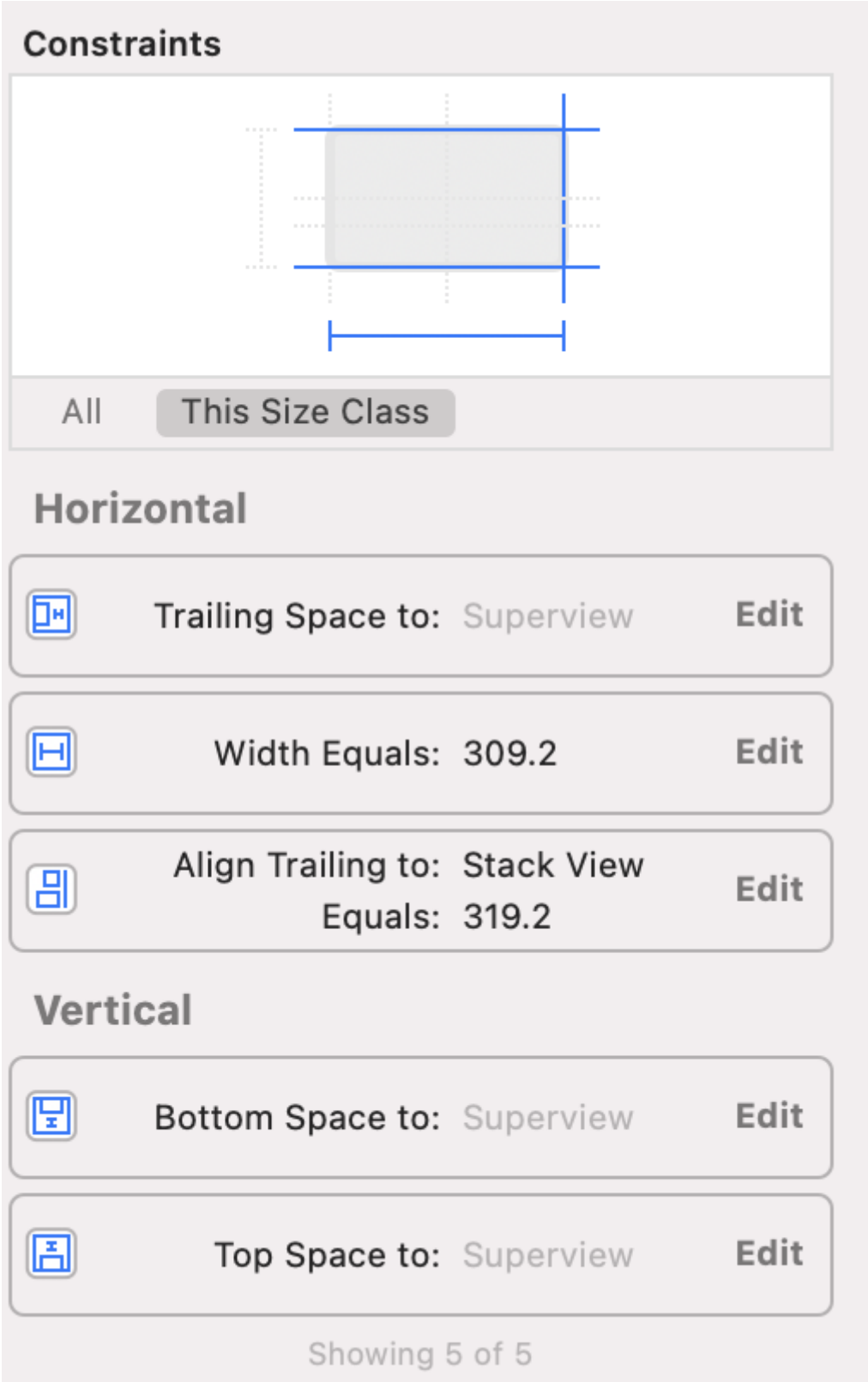
科学计算界面由5行6列按键构成。每一行6个按键组成一个横向排列、均匀填充的StackView，5个行StackView竖向排列组成一个完整的StackView。



科学计算界面的约束有5个——它仅仅在横屏Installed，因此不需要设计竖屏的约束。开始时，我试图仅添加3个针对SuperView，即父视图的约束，控制它完全占据计算界面的左侧，再通过让代表计算界面的StackView采用Fill Proportionally的Distribution使得两个小子计算界面按宽度均匀分布。但仅仅这样做之后，两个子计算界面并没有填满父视图，且科学计算界面的大小被压缩了。于是，我经过计算，给科学计算界面加上宽度约束，并让它和基本计算界面保持适当的距离。最终显示正常。

基本计算界面

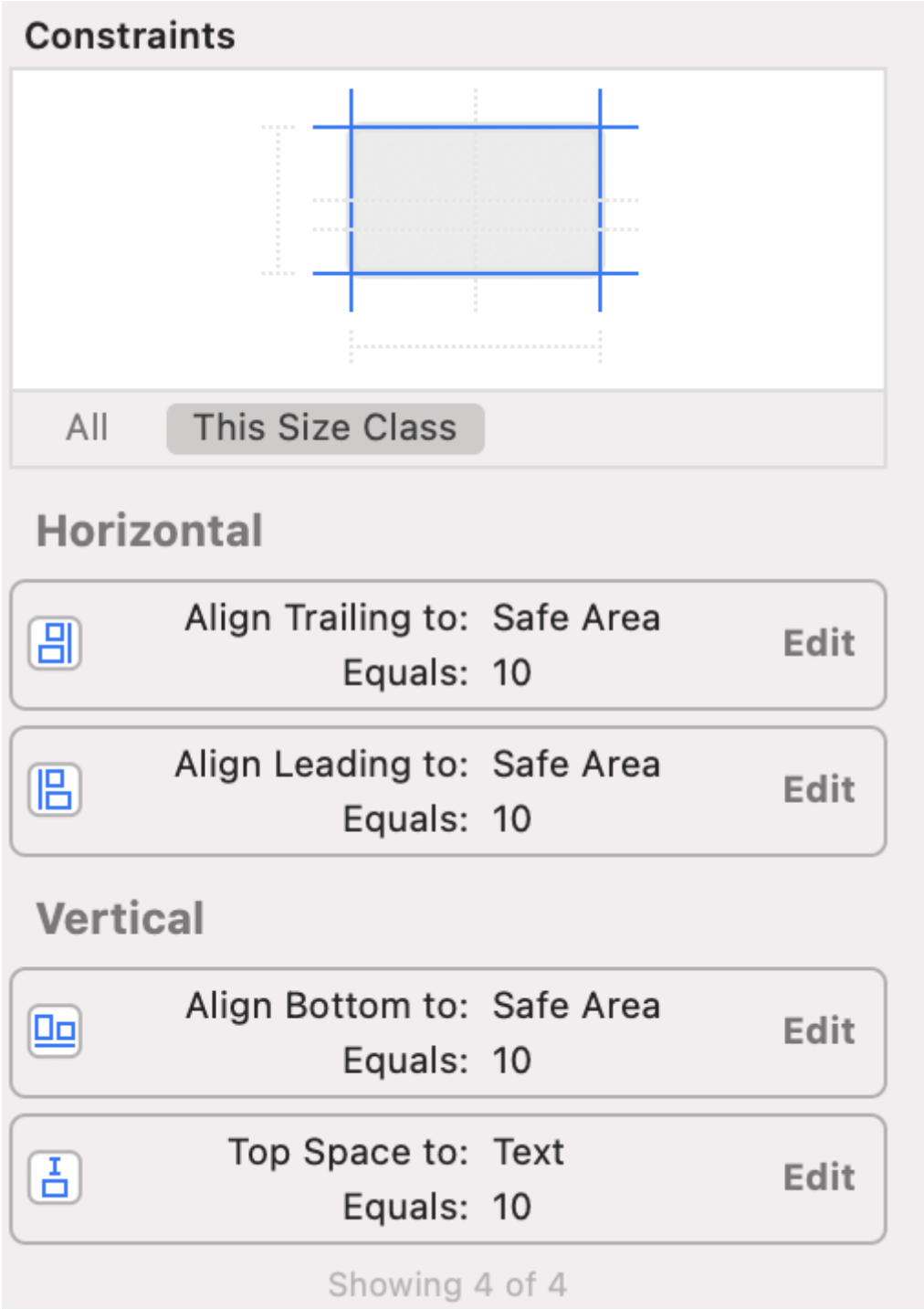
基本计算界面由5行4列按键构成，组合方式与科学计算界面一样。



类似于科学计算界面，我也给它施加了三个针对父视图的约束、1个限制宽度的约束和一个控制到科学计算界面距离的约束。

界面切换

我本来计划通过设置科学计算界面的Installed值来完成横竖屏界面的设计，但发现若令基本计算界面在横竖屏均处于Installed状态，一些横屏的约束不会生效，因此另外复制一份基本计算界面，给其加以不同的约束，让它仅在竖屏显示。这样，界面的显示正常了。



三个界面完成后，横屏基本设计完成。竖屏增删的约束在上文也有提及，不再赘述。

实现计算功能

数字的输入和单元、二元计算均在框架代码中实现，大部分按键仅仅是填充框架，不再赘述。本节主要介绍三个对框架进行改动的功能，角弧度切换、极大极小值处理以及随机数生成。

角弧度切换

同时支持角度和弧度要求计算器拥有状态，这一点容易实现。但是operation这个闭包数组直接通过等号初始化，因此无法在其中的函数引用Calculator类自身的成员变量。所以，另设一个初始化函数，在其中完成涉及角弧度计算的按键功能。

```
146     public func Init(){
147         operations["sin"]=Operation.UnaryOp{
148             op in
149             return sin(Double(op)*self.radMode)
150         }
151         operations["cos"]=Operation.UnaryOp{
152             op in
153             return cos(Double(op)*self.radMode)
154         }
155         operations["tan"]=Operation.UnaryOp{
156             op in
157             return tan(Double(op)*self.radMode)
158         }
159     }
```

至于这个初始化函数，则在ViewDidLoad时调用。

```
22     override func viewDidLoad() {
23         super.viewDidLoad()
24         calculator.Init()
25         // Do any additional setup after loading the view.
26     }
```

状态直接由一个Double类型的成员变量描述，它在按下Rad/Deg键时改变，作为乘以输入的角度/弧度值的系数。

```
case .Rad:
    radMode=1
    return nil
case .Deg:
    radMode=Double.pi/180
    return nil
```

另外，也需在响应函数里进行特殊判断，修改按键的titleLabel。

极大极小值处理

过小的数和过大的数应当给予处理，因为它很可能是某些错误计算的结果。为此，另设一个属性result作为中间变量，存储单元和双元计算的结果，在其set访问器中对得到的数字大小进行判断，若太接近0则返回0，太大则返回nil。

```
var result:Double?{
    get{
        return _result
    }
    set(v){
        if(abs(v!)<1e-6){
            print("to small")
            _result=0
        }else if(v!>2147483647.0){
            _result=nil
        }else{
            _result=v
        }
    }
}
```

为了配合result返回的空值，响应按键按下的函数也要进行修改。若在按下等号时result为0，则输出nan。

```
    @IBAction func onOperationPressed(_ sender: UIButton) {
        if(sender.currentTitle=="Rand"){
            digitsOnDisplay=String(drand48())
            return
        }
        var op=sender.currentTitle  2 ⚠ Variable 'op' was never mutated; consider cha
        var result:Double?=calculator.performOperation(operation: op!,
            operand: (Double)(digitsOnDisplay)!)
                if(result==nil){
                    if(op=="="){
                        digitsOnDisplay="nan"
                    }
                }else{
                    digitsOnDisplay=String(result!)
                }
            inTypingMode=false
        //状态键
        if(sender.currentTitle=="Rad"){
            sender.setTitle("Deg",for: UIControl.State.normal)
        }else if(sender.currentTitle=="Deg"){
            sender.setTitle("Rad",for: UIControl.State.normal)
        }

    }
```

随机数生成

开始时，我打算将随机数当做Constant来生成，但这样一来只会调用一次随机数生成函数，因此通过在响应函数中特殊判断完成这一功能。

```
if(sender.currentTitle=="Rand"){
    digitsOnDisplay=String(drand48())
    return
}
```

总结反思

Xcode的排版功能相当复杂，基本方式却很单纯——仅仅提供了几个属性，因此我上手时遇见困难，好在文档可说是详细，虽然没能直接解决我的问题，但带给了我许多思路。另一方面，Xcode多变的界面、四处隐藏的按钮也给我带来了一定的困惑，相信在以后会慢慢适应。