实验编号： 1 **四川师大《IOS》实验报告 2018** 年 **9** 月 **5** 日

### **计算机科学学院** 级 4 班 实验名称： Swift类和子类 \_

姓名：\_\_\_蒋宇童\_\_\_\_\_\_ 学号：\_\_2016110419\_\_\_\_\_\_\_\_ 指导老师：\_\_李贵洋\_\_ 实验成绩:\_\_\_\_\_

**实验\_一\_ \_\_\_\_\_\_** Swift类和子类**\_\_\_\_\_\_\_\_**

1. 实验目的及要求
2. 用 Swift语言编写类和子类
3. 掌握类和子类的定义及使用
4. 掌握git的基本使用命令
5. 实验内容

* 作业1:（重点：闭包）

1. 找出从1-10000的所有质数，将找出的质数存入可变数组，对数组进行正反排序并输出结果(采用5种以上的方法实现)；
2. 掌握版本控制git的基本使用
   1. git init 初始化版本库
   2. git add . 添加当前目录下的所有文件到暂存区
   3. git commit -m 提交暂存区的文件到版本库

* 作业2:（枚举、类、派生）

1. 实现Person类：
   1. 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；
   2. 具有指定构造函数和便利构造函数；
   3. 两个Person实例对象可以用==和!=进行比较；
   4. Person实例可以直接用print输出；
2. 从Person分别派生Teacher类和Student类：
   1. Teacher类增加属性title，实例可以直接用print输出；
   2. Student类增加属性stuNo，实例可以直接用print输出；
3. 分别构造多个Person、Teacher和Student对象，并将这些对象存入同一个数组中；
4. 对数组执行以下要求：
   1. 分别统计Person、Teacher和Student对象的个数并放入一字典中，统计完后输出字典内容；
   2. 对数组按以下要求排序并输出：age、fullName、gender+age；
5. 实验主要流程、基本操作或核心代码、算法片段（该部分如不够填写，请另加附页）

* 作业1:（重点：闭包）

1. 找出从1-10000的所有质数，将找出的质数存入可变数组，对数组进行正反排序并输出结果(采用5种以上的方法实现)；

* 程序代码：

//: Playground - noun: a place where people can play

import UIKit

var primes=[Int]()

for i in 2...1000

{

var isPrime = true

for j in 2..<i

{

if i%j==0

{

isPrime=false

}

}

if isPrime

{

primes.append(i)

}

}

//从大到小输出

//方法一 使用外部写函数然后调用

var primestwo=primes

func compare(a:Int,b:Int)->Bool

{

return a>b

}

primestwo.sort(by: compare)

//方法二 闭包完整的

var primesthree=primes

primesthree.sort(by: {(num1:Int,num2:Int)->Bool in return num1>num2})

//方法三 闭包 不指定参数参数类型

var primesfour=primes

primesfour.sort(by: {(num1,num2)->Bool in return num1>num2})

//方法四 闭包不指定参数

var primesfive=primes

primesfive.sort(by: {return $0>$1})

//方法五 闭包省略return

var primesfsix=primes

primesfsix.sort(by: {$0>$1})

//方法六 如果闭包是函数调用的最后一个参数，可以将闭包放到括号外面，提高代码的可读性

var primesseven=primes

primesseven.sort(){$0>$1}

//输出

print(primes)

print(primestwo)

print(primesthree)

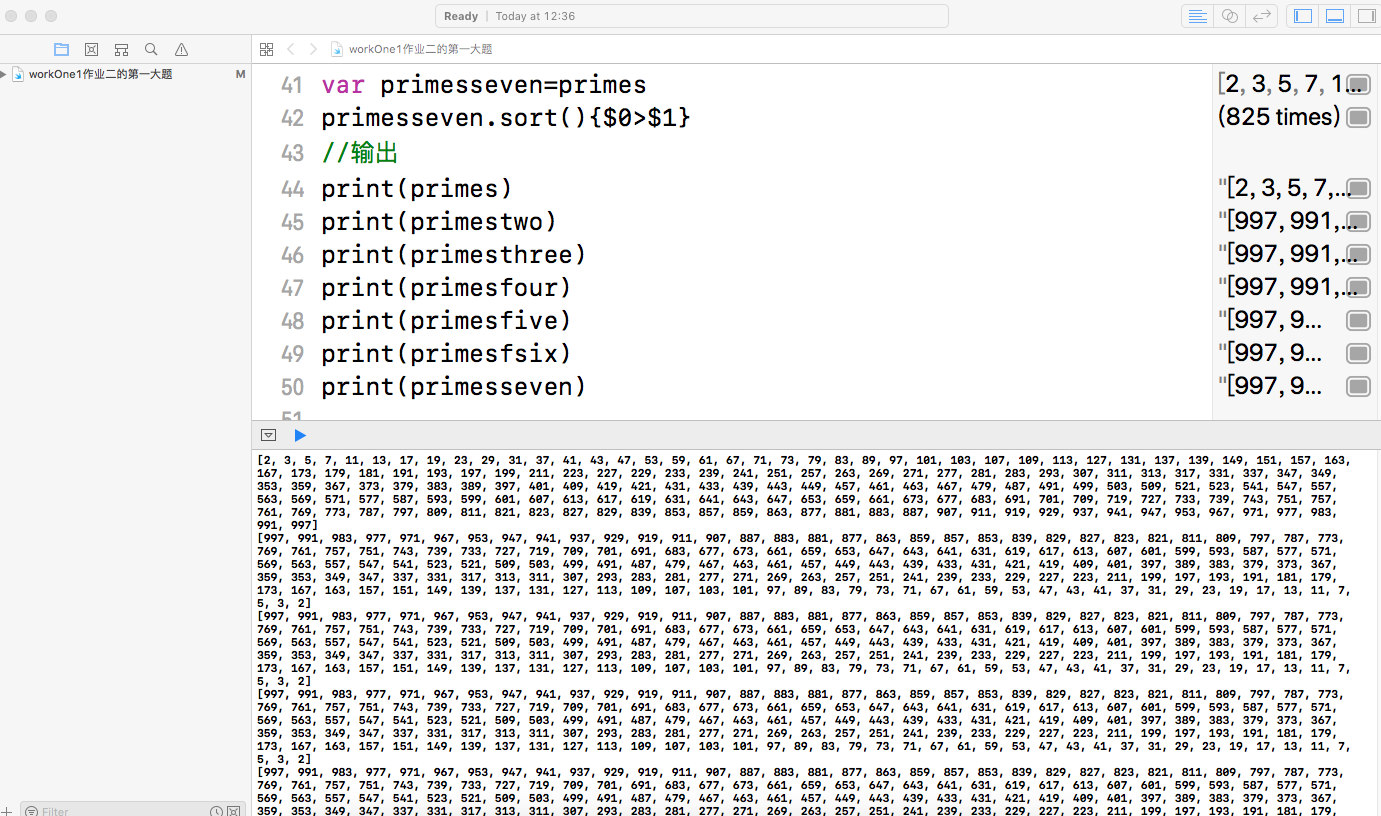
print(primesfour)

print(primesfive)

print(primesfsix)

print(primesseven)

* 运行结果：



1. 掌握版本控制git的基本使用
   1. git init 初始化版本库
   2. git add . 添加当前目录下的所有文件到暂存区
   3. git commit -m 提交暂存区的文件到版本库

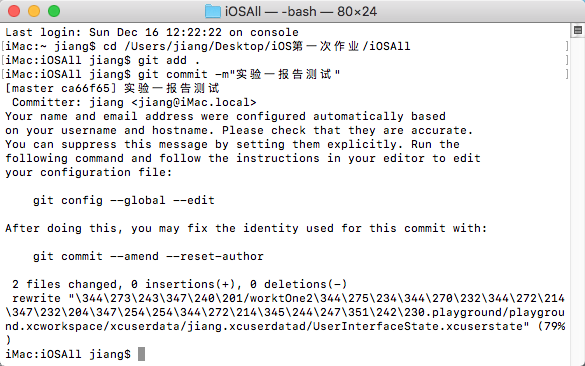
* 程序代码：

git init

git add.

git commit – m” ok”

* 运行结果：



* 作业2:（枚举、类、派生）

1. 实现Person类：
   1. 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；
   2. 具有指定构造函数和便利构造函数；
   3. 两个Person实例对象可以用==和!=进行比较；
   4. Person实例可以直接用print输出；

* 程序代码：

//: Playground - noun: a place where people can play

import UIKit

/\*

（1） 实现Person类：

a) 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；

b) 具有指定构造函数和便利构造函数；

c) 两个Person实例对象可以用==和!=进行比较；

d) Person实例可以直接用print输出；

\*/

enum Gender:Int {

case male,female

static func >(gender1:Gender,gender2:Gender)->Bool

{

return gender1.rawValue>gender2.rawValue

}

}

class Person: CustomStringConvertible

{

var firstName:String = ""

var lastName:String = ""

var age:Int = 0

var gender:Gender

var fullName:String

{

get

{

return self.firstName+" "+self.lastName

}

}

init(firstName:String, lastName:String,age:Int,gender:Gender)

{

self.firstName=firstName

self.lastName=lastName

self.age=age

self.gender=gender

}

convenience init(name:String,age:Int)

{

self.init(firstName:name,lastName:"",age:age,gender:Gender.female)

}

static func ==(p1:Person,p2:Person)->Bool

{

return p1.fullName==p2.fullName&&p1.age==p2.age

}

static func != (p1:Person,p2:Person)->Bool

{

return p1.fullName != p2.fullName||p1.age != p2.age

}

var description: String {

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender)"

}

}

let person1=Person(name: "jyt",age:10);

let person2=Person(name: "jyt",age:20);

let p3=person2

if person1 == person2

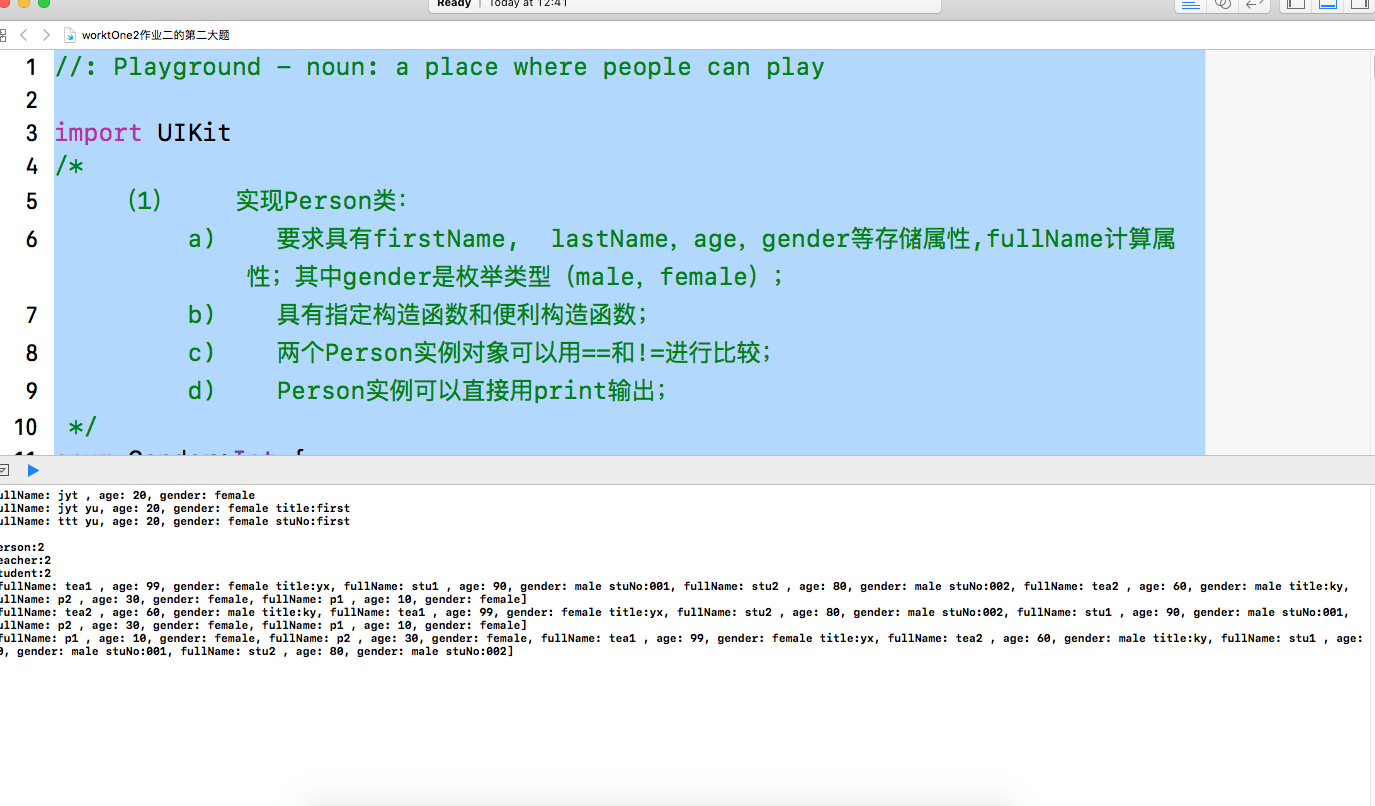
{

print("same")

}

print(p3)

* 运行结果：



1. 从Person分别派生Teacher类和Student类：
   1. Teacher类增加属性title，实例可以直接用print输出；
   2. Student类增加属性stuNo，实例可以直接用print输出；
2. 分别构造多个Person、Teacher和Student对象，并将这些对象存入同一个数组中；
3. 对数组执行以下要求：
   1. 分别统计Person、Teacher和Student对象的个数并放入一字典中，统计完后输出字典内容；
   2. 对数组按以下要求排序并输出：age、fullName、gender+age；

* 程序代码：

//: Playground - noun: a place where people can play

import UIKit

/\*

（1） 实现Person类：

a) 要求具有firstName, lastName，age，gender等存储属性,fullName计算属性；其中gender是枚举类型（male，female）；

b) 具有指定构造函数和便利构造函数；

c) 两个Person实例对象可以用==和!=进行比较；

d) Person实例可以直接用print输出；

\*/

enum Gender:Int {

case male,female

static func >(gender1:Gender,gender2:Gender)->Bool

{

return gender1.rawValue>gender2.rawValue

}

}

class Person: CustomStringConvertible

{

var firstName:String = ""

var lastName:String = ""

var age:Int = 0

var gender:Gender

var fullName:String

{

get

{

return self.firstName+" "+self.lastName

}

}

init(firstName:String, lastName:String,age:Int,gender:Gender)

{

self.firstName=firstName

self.lastName=lastName

self.age=age

self.gender=gender

}

convenience init(name:String,age:Int)

{

self.init(firstName:name,lastName:"",age:age,gender:Gender.female)

}

static func ==(p1:Person,p2:Person)->Bool

{

return p1.fullName==p2.fullName&&p1.age==p2.age

}

static func != (p1:Person,p2:Person)->Bool

{

return p1.fullName != p2.fullName||p1.age != p2.age

}

var description: String {

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender)"

}

}

let person1=Person(name: "jyt",age:10);

let person2=Person(name: "jyt",age:20);

let p3=person2

if person1 == person2

{

print("same")

}

print(p3)

/\*

（2） 从Person分别派生Teacher类和Student类：

a) Teacher类增加属性title，实例可以直接用print输出；

b) Student类增加属性stuNo，实例可以直接用print输出；

\*/

//老师

class Teacher:Person

{

var title:String

init(firstName: String, lastName: String, age: Int, gender: Gender,title:String) {

self.title=title

super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)

}

override var description: String

{

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) title:\(self.title)"

}

}

var tea11=Teacher(firstName: "jyt", lastName: "yu", age: 20, gender: Gender.female, title: "first")

print(tea11)

//学生

class Student:Person

{

var stuNo:String

init(firstName: String, lastName: String, age: Int, gender: Gender,stuNo:String) {

self.stuNo=stuNo

super.init(firstName: firstName, lastName: lastName, age: age, gender: gender)

}

override var description: String

{

return "fullName: \(self.fullName), age: \(self.age), gender: \(self.gender) stuNo:\(self.stuNo)"

}

}

var stu11=Student(firstName: "ttt", lastName: "yu", age: 20, gender: Gender.female, stuNo: "first")

print(stu11)

//（3） 分别构造多个Person、Teacher和Student对象，并将这些对象存入同一个数组中；

var p1=Person(firstName: "p1", lastName: "", age: 10, gender: Gender.female)

var p2=Person(firstName: "p2", lastName: "", age: 30, gender: Gender.female)

var tea1=Teacher(firstName: "tea1", lastName: "", age: 99, gender: Gender.female, title: "yx")

var tea2=Teacher(firstName: "tea2", lastName: "", age: 60, gender: Gender.male, title: "ky")

var stu1=Student(firstName: "stu1", lastName: "", age: 90, gender: Gender.male, stuNo: "001")

var stu2=Student(firstName: "stu2", lastName: "", age: 80, gender: Gender.male, stuNo: "002")

var arry=[p1,p2,tea1,tea2,stu1,stu2]

/\*

（4） 对数组执行以下要求：

a) 分别统计Person、Teacher和Student对象的个数并放入一字典中，统计完后输出字典内容；

b) 对数组按以下要求排序并输出：age、fullName、gender+age；

\*/

var dictionary=["person":0,"teacher":0,"student":0]

var k=dictionary["person"]

print(k!)

for i in arry

{

if i is Teacher {dictionary["teacher"]! += 1}

else if i is Student{dictionary["student"]! += 1}

else { dictionary["person"]! += 1}

}

for (key,value) in dictionary

{

print("\(key):\(value)")

}

var arryTwo=arry

arryTwo.sort{$0.age>$1.age}

print(arryTwo)

var arryThree=arry

arryThree.sort{$0.fullName>$1.fullName}

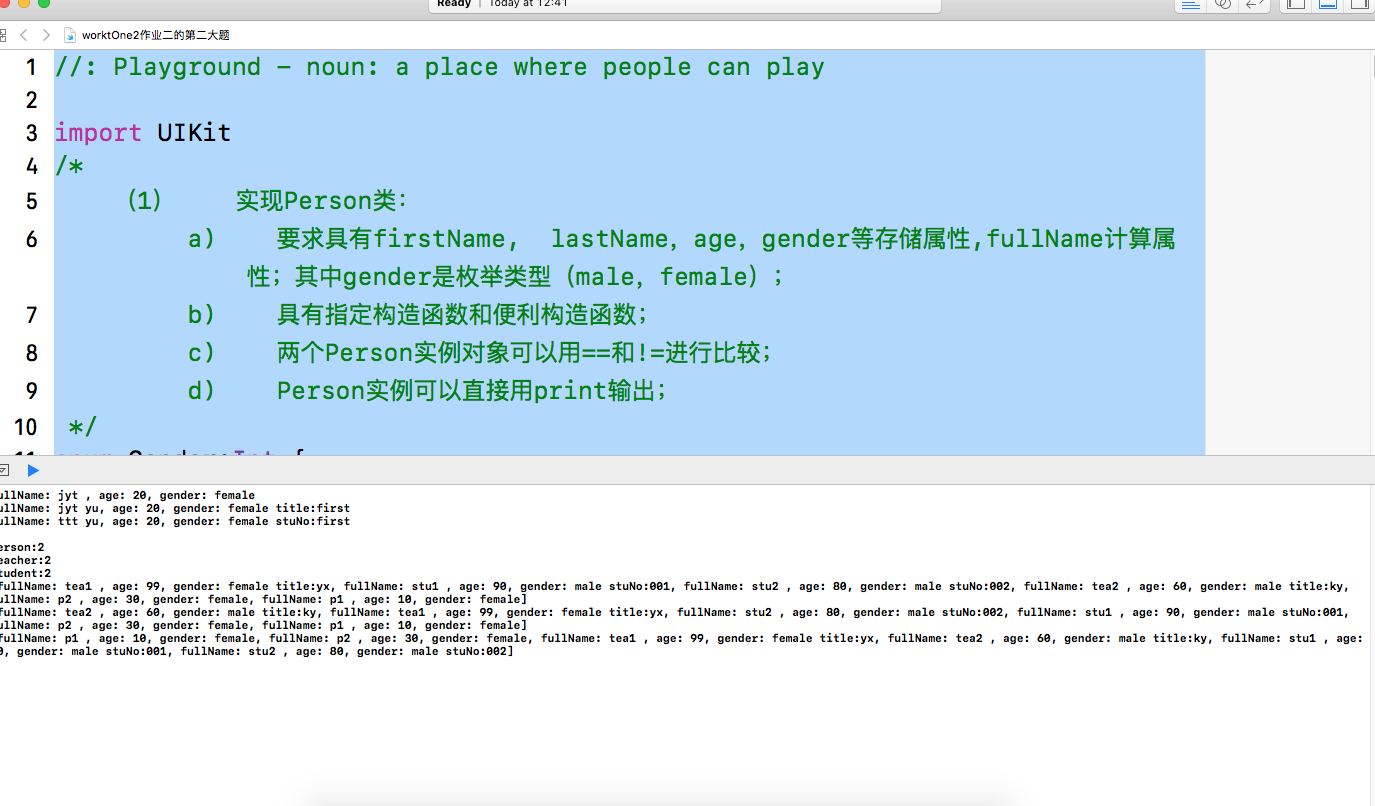
print(arryThree)

var arryFour=arry

arryFour.sort{$0.gender>$1.gender&&$0.age>$1.age}

print(arryFour)

* 运行结果：



1. 实验结果的分析与评价（该部分如不够填写，请另加附页）

**Github：https://github.com/jiangyutong/swiftWork/tree/master/代码**

通过这个实验我学会怎么用闭包，怎么用简单的用git,怎么定义一个类，怎么继承和派生，当我们想输出一个类的时候我们需要重载description函数，我买就可以让类通过print来输出了。我感觉到其实闭包是很主要到，他可以让我们少很多不必要的类名。并且swift中把闭包做的很简化。

注：实验成绩等级分为（90－100分）优，（80－89分）良，(70-79分)中，（60－69分）及格，（59分）不及格。