**import** UIKit

**let** arrayOne = Array<Int>()

**let** arrayTwo = [Int]()

**var** arrayThree: [Int] = []

//3 варианта написания пустого массива

**let** arrayFour = [1, 2, 3, 4, 5]

//обычный массив

**let** arrayFive = [Int](repeating: 10, count: 6)

//массив из 6 элентов со значением 10

arrayThree += arrayFive

arrayThree

// были константы arrayThree - пустой и arrayFive  [10, 10, 10, 10, 10, 10] мы массиву3 присвоили значение массива5, но перед эти сделал массив3 переменной var

arrayFour[2]

//достали из массив4 элемент под индексом 2

arrayThree[1...3] = [15]

//массив3 поменяли массив с 1 по 3 индекс теперь значения 15 и всего тепень не 6 значений, а 4 т.к индекс 3 максимум

arrayThree.count - 1

//count показывает сколько элементов в массиве

arrayThree.append(100)

//добавили в конец массива элемент со значение 100

arrayThree.insert(105, at: 2)

//добавили в массив элемент со значением 105 на место индекса 2 (3 элемент)

arrayThree.remove(at: 4)

//удалить элемент по индексу 4

arrayThree

arrayThree.removeFirst()

arrayThree

//удалили 1-ый элемент

arrayThree.removeLast()

arrayThree

//удалили последний элемент

//СЛОВАРИ

**let** dictOne = Dictionary<String, String>()

**let** dictTwo = [String: String]()

**let** dictThree: [String: String] = [:]

//3 варианта написания пустого словаря

**var** namesAges = ["Ivan": 30, "Vitalik": 30, "Sasha": 25]

namesAges.count

//count показывает сколько элементов в словаре

namesAges.isEmpty

//isEmpty показывает пуст

ой словарь или нет (true или false)

namesAges["Ivan"] = 35

//в словаре мы обращаемся к значению не по индексу, а по ключу

**let** deletedAge = namesAges.updateValue(40, forKey: "Ivan")

namesAges

//мы создали новую констатну и заменили в ней значение ключа Ivan и теперь мы можем работать со старым значение 35 и новым 40

namesAges["Ivan"] = **nil**

//Удаление ключа Ivan

namesAges

**let** deletedValue = namesAges.removeValue(forKey: "Sasha")

namesAges

//мы создали новую констатнту и теперь можем работать с удаленным значением

namesAges = [:]

//теперь мы из словаря с данными сделали из него пустой массив

//Сеты

**let** setOne = Set<String>()

**let** setTwo: Set<String> = []

// Сеты пустые

**var** setThree: Set = [1, 2, 3, 4]

setThree.insert(5)

setThree.insert(6)

setThree.insert(7)

// .insert добавление элемента в сет

setThree

setThree.isEmpty

//пустой сет или нет

setThree.count

//количество элементов в сете

setThree.remove(6)

setThree

// удаление элемента со значение 6

setThree.contains(1)

// проверка есть ли этот элемент в сете или нет тру или фолс

**let** setFromIneToThree: Set = [1, 2, 3, 4]

**let** setFromFourToNine: Set = [1, 2, 4, 5, 6, 7, 8, 9]

**let** allValueArray = setFromIneToThree.union(setFromFourToNine).sorted()

// мы создали для сета и объединили их в один с помощью оператора .union и отсортировали с помощью .sorted

**let** commonValueSet = setFromIneToThree.intersection(setFromFourToNine)

// с помощью оператора .intersection мы можешь проверить  содержится одинаковые элементы в двух сетах

**let** notRepeatedValuesArray = setFromIneToThree.symmetricDifference(setFromFourToNine).sorted()

// возвращаем массив занчений, которые не повторяются с помощью оператора .symmetricDifference и отсортировали их с помощью .sorted

**let**  substractedValuesArray = setFromIneToThree.subtracting(setFromFourToNine).sorted()

// записать значения сета setFroneIneToThree , которые не повторяются во втором сете setFromFourToNine с помощью оператора .subtracting и отсортировали с помощью .sorted

// УСЛОВИЯ if

**let** a = 3

**let** b = 1

**if** a==b {

 print("a ровно b")

} **else** **if** a < b {

     print("b больше a на столько \(b - a)")

} **else** **if** a > b{

    print("a больше b на столько \(a - b)")

}

**let** isSunny = **true**

**var** action = ""

**if** isSunny {

    action = "я пойду погуляю"

} **else** {

    action = "я буду сидеть дома если не солнечно"

}

//здесь мы написали функци, которая говрит если солнечно я пойду гулять если нет то нет

action = isSunny ? "я пойду погуляю" : "я буду сидеть дома если не солнечно"

// с помощью тернарного оператор ? можно функцию выше сократить в одну строчку

/\*

еще функции операторы можно использовать внутри конструкции if

 if a != b

 if a <= b

 if a >= b

 if a > b

 if a < b

 if ~=    посмотреть принадлежить ли данное число диапазону пример ниже

  \*/

**if** 1...4 ~= b {

    print("Hello World!")

}

// если значение b входит в диапазо от 1 до 4 ( у нас b=1) , то мы выводим запись Hello World!

**let** c = 5

**if** c == 2 && a == 10 || c == 5 {

    print("Hello coder")

} **else** {

    print("NO")

}

// составное условие если с = 2 и а = 10 или с = 5, то выводится записть Hello coder иначе выведется запись NO

**let** names = ["Jordan", "Alexa", "Siri", "Alice"]

**let** name = names[1]

print("Best wishes, \(name)!")

//Инструкция guard

**func** someFunc(a: Int, b: Int) {

**guard** a == b **else** { **return** }

  // если а = b то код продолжается если нет то 1 итерация прекращается

**guard** b == 11 **else** { **return** }

  // sесли а = 11 то код продолжается если нет то 1 итерация прекращается

**if** a == b {

    // some code to run

**if** b == 11 {

    // тоже самое что и выше но только в if

    }

  }

}

**for** i **in** 1...5 {

**guard** i != 3 **else** { **break** }

  print(i)

}

//записали цикл числа в диапазоне от 1 до 5 если i не равно 3 то код продолжается.  Если бы в {continue} ,то код пропусти 1 итерацию (3) и код продолжится

//ответ 1 2 4 5

// Если в фигурных скобках {break}, то код вообще прекращается

//ответ 1 2

**for** i **in** 1...5 {

**if** i == 3 {

**break**

  }

  print(i)

}

//тот же цикл но с оператором if

//guard хорош тем ,что можем записать несколько строчек подряд с гуардом и выполнение кода будет продолжаться уже под этими строчками. т.е. условия выполнены

//switch - переключатель с множества вариантов ( альтернатива условий if)

**let** totalScore = 20

**if** totalScore == 10 {

  print("you're not jedi")

} **else** **if** totalScore == 20 {

  print("you're still not jedi")

} **else** **if** totalScore == 100 {

  print("you're jedi")

}

// это запись обычного условия

// тоже самое мы можем записать  switch

**switch** totalScore {

**case** 10, 20:

  print("you're not jedi")

**fallthrough**

    // чтобы провалиться в следующий кейс пишем слово fallthrough. в ответе получается будет ответ в том кейсе что ты написал фаллфру

    // и следующие кейсы

**case** 50..<100:

  print("you're almost jedi")

**case** 100:

  print("you're jedi")

**default**:

**break**

}

// чтобы case прекратился пишем в конце

//default:

//break

//смысл свитча что он проверяет все возможные варианты событый. и он выполняется быстрее , тк код проверяет только кейсы,а не весь код

// как в if

//

//for in циклы

Укажите, что будет выведено на экран в результате выполнения данной программы.

//Циклы While и repeat while

//var timer = 5     делаем таймер от 5 до 1 и старт

//print("Couting down")

//

//while timer > 0 {       условие таймер больше 0 то выполняется

//  print(timer)

//  timer -= 1             с каждым шагом - 1

//}

//

//print("Start!")

// тоже самое только репид вайл - т.е если условия прекратятся выполняться то 1 итерация выполнится

**var** timer = 5

print("Couting down")

**repeat** {

  print(timer)

  timer -= 1

} **while** timer > 0

print("Start!")

//ФУНКЦИИ

// 1. простая функция, ничего не принимающая и ничего не возвращающая

**func** sayHello() {

  print("Hello")

}

sayHello()

// 2. функция, принимающая один параметр

**func** oneParam(param: Int) {

**var** param = param

  param += 1

}

oneParam(param: 11)

// 3. функция, не принимающая параметров, но возвращающая значение

**func** returnValue() -> Int {

**return** 10

}

**let** a = returnValue()

a

// 4. функция, принимающая несколько параметров и возвращающая значение

**func** giveMeYour(name: String, andSecondName: String) -> String {

**return** "Your full name is \(name) \(andSecondName)"

}

giveMeYour(name: "Ivan", andSecondName: "Akulov")

// 5. функция, принимающая массив в качестве параметра и использующая вложенную функцию для работы

**func** calcMoneyIn(array: [Int]) -> Int {

**var** sum = 0

**func** sayMoney() {

    print(sum)

  }

**for** item **in** array {

    sum += item

  }

  sayMoney()

**return** sum

}

calcMoneyIn(array: [1, 2, 3, 4, 5])

// 6. функция, которая принимает переменное число параметров

**func** findSum(ofIntegers integers: Int...) -> Int {

  // в скобках функции указан первое и второе именя входного параметра

**var** sum = 0

**for** item **in** integers {

    sum += item

  }

**return** sum

}

findSum(ofIntegers: 1, 2, 3, 4, 5, 6)

// 7. имена параметров функции (если мы не ходим задавать взодному парметру имя можно просто указать нижнее подчеркивание \_)

**func** sum(\_: Int) -> Int {

**return** 10

}

// 8. функция в качестве возвращаемого значения

**func** whatToDo(missed: Bool) -> (Int) -> Int {

**func** missCountUp(input: Int) -> Int { **return** input + 1 }

**func** missCountDown(input: Int) -> Int { **return** input - 1 }

**return** missed ? missCountUp : missCountDown

}

**var** missedCount = 0

missedCount = whatToDo(missed: **true**)(missedCount)

missedCount = whatToDo(missed: **false**)(missedCount)

//ЗАмыкание это те же функции но без имени

//let closer = {

//    print("Hello, world!")

//}

//вверхнюю часть кода можем убрать с помощью замыкания.

**func** repeatThreeTimes(closer: () -> ()){

**for** \_ **in** 0...2{

        closer()

    }

}

//repeatThreeTimes(closer: () -> ())

//если убрать вверхнюю часть кода и эту и оставить не закомментирование результат будет тот же

repeatThreeTimes(closer: {

    print("Hello, world!")

})

**let** unsortedArray = [123, 2, 32, 67, 8797, 432]

**let** sortedArray = unsortedArray.sorted {

    (number1:Int, number2:Int) -> Bool **in**

**return** number1 > number2

}

//здесь с помощью замыкания мы отсортировали массив чисел от большего к меньшему

//КОРТЕЖИ

//пример0

**let** one = 1

**let** two = 2

**let** three = 3

(one, two, three)

//пример1

**let** boy = (5, "Sergey")

boy.0

boy.1

//пример2

**let** (first, second, third) = (1, 2, 3)

first

second

//пример 3

//мы записали зарактеристики зеленного карандаша и в дальшнейшем можем обращаться к каждой характеристике

**let** greenPencil = (color: "green", length: 20, weight: 4)

greenPencil.color

greenPencil.length

greenPencil.weight

**let** (greenColor, greenLength, greenWeight) = greenPencil

//пример4

//мы имеем массив даных имя-возраст. Мы хотим определить самого старшего человека

**let** agesAndName = ["Misha": 29, "Kostya": 90, "Mira": 30]

**var** age = 0

**var** name = ""

**for** (nameInD, ageInD) **in** agesAndName {

**if** age < ageInD {

        age = ageInD

        name = nameInD

    }

}

age

name

//ОПЦИОНАЛЫ

**var** fuel: Int?

fuel = 20

//fuel = nil

//print("\(fuel!) liters left")

// мы объявили переменную фуел (топливо) опциональным т.е значение может быть, а может не быть. ниже создали принт и в скобках сделаи интерпалацию строки значения фуел. если у нас топливо ровно 20, то в ответе будет слово optional

//чтобы его не было можно после интерполяции переменной поставить знак !, но если значение nil то будет ошибка

//познакомимся с опциональной привязкой. изъятие значения. если значение nill то ошибка

**if** **let** availableFuel = fuel {

    print("\(availableFuel) liters left")

} **else** {

    print("no feul data available")

}

//guard

**func** checkFuel() {

**guard** **let** availableFuel = fuel **else** {

        print("no feul data available")

**return**

    }

    print("\(availableFuel) liters left")

}

checkFuel()

**import** UIKit

//Классы

//шаблон по которому создаются объекты

//ключевое слово class

//все изменяемые параметры пишутся через переменную var

//внутри классов есть функции называются они методами, но пишется func

**class** Human {

**var** name = "Ivan"

**var** age: Int? = 30

    // возраст может быть, а может не быть

**var** hairs = **true**

  //метод-функция возращает строку

**func** description() -> String {

**if** **let** humanAge = age {

**return** "Hello! My name is \(name) and I'am \(humanAge) years old!"

    } **else** {

**return** "Hello! My name is \(name)!"

    }

  }//если есть возраст то выдаст 1 строку если нет то вторую

}

//экземпляры класса

**var** humanOne = Human()

humanOne.name = "Natasha"

humanOne.name

humanOne.description()

//у этого экземпляра есть это свойство human и его свойства

**let** humanTwo = Human()

humanTwo.hairs = **false**

humanTwo.name = "Jack"

humanTwo

//humanOne = humanTwo выдаст ошибку потому что мы не может приравнять два let константы

//humanOne.name

**var** array = [Human]()

array.append(humanOne)

array.append(humanTwo)

//если создаем массив и поестить туда 2 экземпляра обязательно [Human] инициализируем класс

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

**import** UIKit

//инициализаторы класса

//Функции или методы которые задают изначальные значения параметров класса

//создали класс human

**class** Human {

**var** name: String

**var** age: Int?

**var** hairs: Bool

**func** description() -> String {

**if** **let** humanAge = age {

**return** "Hello! My name is \(name) and I'am \(humanAge) years old!"

    } **else** {

**return** "Hello! My name is \(name)!"

    }

  }

  //т.к мы параметрах класса name - string  и hairs - bool изначально нет параметров

    // мы их зададим с помощью ключевого слова init

**init**() {

    name = "Ivan"

    hairs = **true**

  }

  // также можно инициализатор задать по другому пишем в одну строку

    // и снизу с помощью ключевого слова self обращаемся к методу

**init**(name: String, age: Int?, hairs: Bool) {

**self**.name = name

**self**.age = age

**self**.hairs = hairs

  }

}

**let** human = Human()

human.age

human.name

human.hairs

**let** human1 = Human(name: "Jason", age: 40, hairs: **true**)

пример

**import** UIKit

**class** Student {

**var** name : String

**var** surname : String

**var** yearOfBorn : Int

**var** mark : Double

**init**(name: String, surname : String, yearOfBorn : Int, mark : Double) {

**self**.name = name

**self**.surname = surname

**self**.yearOfBorn = yearOfBorn

**self**.mark = mark

    }

}

**let** studentOne = Student (name: "Ivan", surname: "Petrov", yearOfBorn: 1, mark: 1.0)

**let** studentTwo = Student (name: "Sergey", surname: "Sidorov", yearOfBorn: 2, mark: 2.0)

**let** studentThree = Student (name: "Nikita", surname: "Ivanov", yearOfBorn: 3, mark: 3.0)

**let** studentFour = Student (name: "Artem", surname: "Kozlov", yearOfBorn: 4, mark: 4.0)

    studentOne

    studentTwo

    studentThree

    studentFour

**var** arrayStudent = [Student]()

arrayStudent.append(studentOne)

arrayStudent.append(studentTwo)

arrayStudent.append(studentThree)

arrayStudent.append(studentFour)

arrayStudent

**var** newArrayStudent = arrayStudent.sorted(by: { $0.mark > $1.mark })

newArrayStudent

НАСЛЕДОВАНИЕ

**import** UIKit

//наследование только классы и только от классов

//наследование нужно чтобы не дублировать функцию

**class** Human {

  // Свойства

**var** name: String

  // Метод

**func** tellAboutMe() -> String {

**return** "Hello! My name is \(name)!"

  }

  // Инициализатор

**init**(name: String) {

**self**.name = name

  }

}

//еще один класс которые наследует от human, но мы добавили в готовый класс еще переменную toy

**class** Child: Human {

**var** toy = "Horse"

  //переопределяем метод с помощью override. super это тот класс от которого наш класс наследует.

    //с помощью + добавляем еще текст

    //self обязательно это обращение к классу

**override** **func** tellAboutMe() -> String {

**let** originalText = **super**.tellAboutMe()

**return** originalText + " And I have a toy \(**self**.toy)"

  }

  //инициализатор

    //name в суперклассе , а toy в просто классе

**init**(toy: String, name: String) {

**self**.toy = toy

**super**.init(name: name)

  }

  //инициализатор по другому тоже самое что выше

**override** **init**(name: String) {

**self**.toy = "Hummer"

**super**.init(name: name)

  }

}

// наш новый ребенок

**let** child = Child(name: "Klava")

child.tellAboutMe()

child.name

child.toy

// еще один ребенок

**let** child1 = Child(toy: "Dinosaur", name: "Max")

child1.tellAboutMe()

//пердотвратить наследование с помощью ключа final

//final func tellAboutMe() -> String {

  //return "Hello! My name is \(name)!"

// так же можно перед классом поставить final и так мы будем запрещать наследовать этот класс вообще

//21 Вычисляемые свойства

**import** UIKit

//создаем класс прямоугольник и называем его элементы высота ширина  глубина

**class** Rectangular {

**let** height: Int

**let** width: Int

**let** depth: Int

    //вычисляемое свойство - это всегда переменная

**var** volume: Int {

**return** height \* width \* depth

  }

  // называем инициализатор (начальные значения)

**init**(height: Int, width: Int, depth: Int) {

**self**.height = height

**self**.width = width

**self**.depth = depth

  }

}

//расчет

**let** rect = Rectangular(height: 10, width: 12, depth: 13)

rect.volume

//ответ 1560 обхем куба

//рассмотри еще один класс персона

**class** Person {

**var** name: String

**var** secondName: String

  // вычисляем полное имя

  // мы создаем новую переменную

**var** fullName: String {

**get** {

**return** name + " " + secondName

    }

// слово return нужно т.к мы объявлем что наша переменная

//будет иметь тип стринг вернет значенире типо стринг

//мы хотим поместить  в наш массив 2 элемента с разделением пробел

**set**(anotherNewValue) {

**let** array = anotherNewValue.components(separatedBy: " ")

      name = array[0]

      secondName = array[1]

    }

  }

**init**(name: String, secondName: String) {

**self**.name = name

**self**.secondName = secondName

  }

}

**let** person = Person(name: "Ivan", secondName: "Akulov")

person.fullName

person.name

person.secondName

//присвоили новое значение set - устанавливаем новое значение

//get берем новое значение

person.fullName = "Zinaida Petrova"

person.name

person.secondName

//свойства классов

// представим что завод создает одинаковые машины

// однако машины попадают в разные места и их продают

// использует разные люди

//есть 2 свойства которые никогда не меняются свой вес и максимальный вес

**import** UIKit

**class** Car {

  //properties

**let** products: Int

**let** people: Int

**let** pets: Int

**class** **var** selfWeight: Int { **return** 1500 }// мы возвращаем значение инт со значение 1500

**class** **var** maxWeight: Int { **return** 2000 }//мы возвращаем значение инт со значение 2000

  //тк у нас собственный вес и вес максильмный никогда не меняются

    //можно поместить их в сам класс

  //computed property

**var** totalWeight: Int {

**return** products + people + pets + Car.selfWeight

  }

  //initializer - изначальные значения

**init**(products: Int, people: Int, pets: Int) {

**self**.products = products

**self**.people = people

**self**.pets = pets

  }

}

**let** car = Car(products: 30, people: 500, pets: 50)

**let** maxWeight = Car.maxWeight // максимальный вес

**let** carWeight = Car.selfWeight //собственный вес

**let** totalWeight = car.totalWeight //общий вес авто. car с мелкой буквы т.к это свойство экземляра класса

//тогда теперь мы можем создать условие если для вычисления

**if** maxWeight < totalWeight {

  print("You can't drive because car is overloaded: \(totalWeight - maxWeight)")

} **else** {

  print("You can drive!")

}

//Ленивые свойства

**import** UIKit

// внешнаяя функция, которая выполняет последнюю функцию

**func** bigDataProcessingFunc() -> String {

**return** "very long process"

}

//есть 1 класс с 3 свойствами

// при создании класса инициализируются все 3 свойства

//с помощью ленивых свойств можно помечать с помощью lazy

//и инициализировать потом, когда будет удобно

**class** Processing {

**let** smallDataProcessing = "small data processing"

**let** averageDataProcessing = "average data processing"

**lazy** **var** bigDataProcessing = bigDataProcessingFunc()

}

//3 свойство с lazy получается опциональной и она не может быть константой

// она всегда var

// и по умолчанию nill пока мы не обратимся к ней

**let** process = Processing()

process.bigDataProcessing

process

//ленивым свойством называют такое своейство которое не инициализируются

//пока к нему не обратятся

//и сопровождается ключевым словом lazy

//после того как мы обратились ленивому свйосвтва оно инициализируется

//до тех пор пока не удаляется из памяти пока его не удалят

//Наблюдатели свойства

// наблюдатели - это наблюдатели за изменением значения свойств

**import** UIKit

//класс работник секретной лаборатории

**class** SecretLabEmployee {

//вар потому что это значение будем менять

**var** accessLevel = 0 {

    //будет установлено

**willSet**(newValue) {

      print("new boss is about to come")// вот вот будет новый босс

      print("new access level is \(newValue)")

    }

    //было установлено

**didSet** {

**if** accessLevel > 0 {

        accessToDB = **true**

      } **else** {

        accessToDB = **false**

      }

      print("new boss just come")

      print("last time I had access level \(oldValue)")

    }

  }

//если у сотрудника уровень доступа 0 то по умолчанию ему доступ ко всему закрыт фолс

**var** accessToDB = **false**

}

**let** employee = SecretLabEmployee()

employee.accessLevel

employee.accessToDB

employee.accessLevel = 1

employee.accessToDB

// Алиасы типо (псевдонимы типов)

**import** UIKit

// даем псевдоним лет length

**typealias** Meter = Int

**let** length: Meter = 50

**let** length1 = 20

//теперь можем сложить и ошибки не будет

**let** sum: Meter = length + length1

//так же можем алиас поставтиь для кортежа из 2 чисел

**typealias** DoubleInteger = (Int, Int)

**let** someConstant: DoubleInteger = (1, 2)

//пустой словарик

**typealias** DictionaryType = [String: Int]

**var** dictionary: DictionaryType = [:]

dictionary["Apartment123"] = 123

dictionary

//перечисления и энумы

//Enums-это типо 1-го класса  swift, они объеденяют в себе связанные значения и присваивают им

//один общий тип. таким образом работая с Enums мы не можем выйти за рамки этого типа

//Enums и перечисления

//пример 1

// создаем класс и назыаем его Monement  допустим у нас есть игра и ходить можно

//только вверх вниз влево и право

//case основная составляющая часть enum

//все значения case должны быть индивидуальными (не повторяться)

**enum** Movement: Int {

**case** forward = 10

**case** backward = 23

**case** left = 101

**case** right = 123

}

//создаем константу с одни из параметров enum

**let** movementDirection  = Movement.backward.rawValue

//пример2

//создаем еще один энум с кейсом и 2 вычисляемыми свойствами

**enum** Device {

**case** iPad(color: String), iPhone

  // тут пишем self потому что мы перебераем занчения собсвтенного энума

    //т.к дефолтного значения у нас нет то свитч его и не требует

**var** year: Int {

**switch** **self** {

**case** .iPhone: **return** 2007

**case** .iPad(**let** color) **where** color == "black": **return** 2020

**case** .iPad: **return** 2010

        //добавили еще 1 кейс именно для черного цвета и его отдельный год

    }

  }

}

//обращение к нашему устройству

**let** yearOfProduction = Device.iPad(color: "black").year

//пример 3

// в энум мы можем вкладывать другой энум (вложенный)

// оружие

**enum** Character {

**enum** Weapon: Int {

**case** sword = 4

**case** wand = 1

    // добавляем 1 свойство в энум

**var** damage: Int {

**return** rawValue \* 10

    }

  }

  // еще один вложенный энум

**enum** CharacterType {

**case** knight

**case** mage

  }

}

//посотмреть повреждения от конкретного оружия

**let** charWeapon = Character.Weapon.sword.damage

//пример4

//индиректные энумы

//в один из кейсов мы хотим поместить один из кейсов

//если много кейсов не нужно напротив каждого кейса писать индерект легче

//написать перед энум

**indirect** **enum** Lunch {

**case** salad

**case** soup

**case** meal(Lunch, Lunch)

}

//наш ланч состоит из салата и супа

**let** myLunch = Lunch.meal(.salad, .soup)