Kotlin

Val fish = 2

Fish.times(2) //output 4

.plus

.div

.minus

Fish.ToInt / .toLong()

Var = dim

Val = const

Object warpper: val boxed: Number = 1

Fun \_\_\_(){

}

No ;

Println(“guhgivswoify”)

Dim as as string //null

String as; //null

Kotlin: var as : Int? = null

Var as: List<String>? = listOf(nuill. null)

Var as: List<String>? = null

as!!.eat()

! = bang !!= double bang (what is it(not null

//check whether var is not null

Var num = 5

Num?.dec() ?: 0 //if num not null, use num as return, otherwise return 0

?: //elvis operator

“He has ${ a + b} cows - $a”

== != //comparison

<> //SQL not

If (num in 1..100) do // if number in 1- 100 range

When (input){

In 0..100 -> println //auto break

101 -> println

Else -> println

}

When{

Happy == 100 -> “happy”

}

Val myList =mutableListOf(“”,””,””) //reference

myList = mutableListOf(“”,””) //error

myList.remove(“”) // fine //true

ListOf(5,12) //collection

Val variable = arrayOf(“hi”, 73254672) //unspecified type

Val intArray = intArrayOf(1,2,3)

Println(Arrays.ToString(variable)) //to print => string

Val nestArray = arrayOf(variable, intArray)

val array = Array(6){it \*2} //array size, it = array index (from 0 , +=1)

println(array.asList()) //[0, 2, 4, 6, 8, 10]

for(element in arrayvariable) //for each

for((index,element) in array.withIndex()){} //built-in index function

for (i in ‘b’… ‘d’){println(i)} // bcd (range in alphabet)

for(i in 5 downTo 1 step 2) // 54321 (downTo = step -1)

array[5]

100.mod(7) //2

Fun main(){} //   
// every function return value   
// return "type unit" == no value

Fun main(args: Array<String>){} // Array<String> can be set in Run> Edit Configuration> Program Argu

*println*("Hello ${args[0]}!")

args[0] //applied to nodejs , good

val isUnit = println(“hi”)

println(isUnit) //return Kotlin.Unit

val isHot = if(temp ==20) true else false //false

fun randomd() : String{} // return a string value

fun swim(speed: String = “fast”){} //parameter default fast

swim(speed = “slow”)

//kotlin style variable declaration

fun getPair() = Pair(1, "foo")

fun num() = 20

fun isTooHot(temp :Int) = temp> 20

//slow your app and potential out-of-memory errors

Fun make() = println(“”)

Fun makenew( var :Any = make()) {} //if no parameter passed, whole var is made as default ==make()

Repeat(2){} //kotlin library (same as for loop)

//Cannot assign for/while in variable

// val variable = for(x in 1..2){} // val variable= while(){}

//kotlin (not interchangable)and C#

‘ ’ for char

“ ” for string

//filter == indexOf() // contains

Val variable = listOf(“hey”, “heyman”, “on9”)

Val list\_that\_only\_have\_h = variable.filter{ it[0]} == ‘h’} //it[0] == first character // [hey, heyman] //printable, eager?

Val filter\_lazily = variable.asSequence().filter{ it[0]} == ‘h’} //how API works

Println(filter\_lazily.toList())

Val lazy\_map = variable.asSequence().map { println (it) } // output: hey

Println(lazy\_map.first()) //output:hey

Println(lazy\_map.toList()) // [“hey”, “heyman”, “on9”]

Fun \_\_\_(){} //named function

{ code of lines }() //Lambda //anonymus function or function literals ///high-order function

Val swim = { print(swim) } swim() //lambda

Val lambdafunction = { variable :String -> code of lines} //lambda parameters

Val lambdafunc : (Int) -> Int = {parameter -> return} //lamda only data type

Fun function1( variable :Int ) = variable +10

Fun update(dirty :Int , lambdaVariable :(Int) -> Int): Int{ return lambdaVariable(dirty) }

Var dirty = 20

Dirty = update(dirty, lambdafunc) //parameter Int, lambda

Dirty = update(dirty, ::function1 ) // ::function1 is not call function but parse a reference as lambda

Dirty = update(dirty) { dirty -> dirty +50 } //exactly use lambda behind lol

**val** random1 = random() //value assign at compile (wont change runtime

**val** random2 = {random()} //every time call, execute (as reference)

Classes -

Class Something{

Val height = 50 //properties

Get(){ return height} or get() = height

Public Set (value) { height = value \* 2} //no one outside can access setter

Fun functionName = height

}

Fun another(){ val variable = Something()

Println(“ ${variable.height} , ${variable. functionName}”) }

Class something ( var/val parameter :Int = “default value” ){ //var/val create properties

Constructor( variable2 :Int) :this() {} } //define constructor, calculation specific for constructor

Val variable = something( variable2 = 89)

//most class only specify one constructor with default parameter //constructor overload not needed]

Class something( val number: Int){

Val variable :Int

Init { when(number){ 0 -> 5

In 1..3 -> 7}} //return5

Constructor() : this ( 8 ) {}}

Class something () :Any {} // All Class by default inheritance with Any

Open class something() { //open is necessary for inheritance

Open var variable = 50} // open is necessary for inheritance

Class some2() : something(){ //inheritance by naming the data type as the class

Override var variable = 60} //open is needed before override

Some basic function, properties like: people head = 1

Abstract class and interface => both cant instantiate object

Abstract class has constructor logic but interface doesn’t

Abstract class (with constructor) or Interfaces // both are class that cant instantiate their own = cannot create object

Abstract class Name{ abstract val vaiable :Int }//create abstract class, abstract variable =something must have

Class Name2 : Name () { override val variable = 7 } //make sure it contains in the class