Here are some Kotlin basics that I have tried. It might be useful as a quick reference:

```
var age = 24

// Explicitely Defined Type
var age: Int = 24

==========IMMUTABLE VARIABLES ========
// can be null ..but not quite this way
val str: String = null

// ? defines safe checking
val str: String? = null

str
null

str.length
error: only safe (?.) or non-null asserted (!!.) calls are allowed on
a nullable receiver of type String?

str?.length
null

str!!.length
kotlin.KotlinNullPointerException

val str:String? = "Peter"

str
Peter

str?.length
5

============== IF EXPRESSIONS =============
```

```
val x = if(price > 19){
println("good price")
"good"
} else if( price > 29){
println("best price")
"best"
} else {
println("too high")
"high"
}
============== WHEN EXPRESSIONS ===========
val price = 39

when(price){
1 -> println("Free")
!in 2..39 -> println("Perfect price")
45 -> println("kotlin is fun")
else -> println("Exceptional price")
}
Perfect price

/* with left hand side function */

when(price){
1 -> println("Free")
in 2..39 -> println("Perfect price")
10+20 -> println("kotlin is fun")
else -> println("Exceptional price")
}
Perfect price

/* Expressions */

val x = when(price){
1 -> println("Free")
in 2..29 -> println("Perfect price")
10+20 -> println("kotlin is fun")
else -> println("Exceptional price")
```

```
}
kotlin is fun

x
kotlin.Unit

/* left hand side function */

when{
price <= 19 -> println("Free")
price <= 29 -> println("Perfect price")
else -> println("Exceptional price")
}
Exceptional price

price
30

val price = 20

/* function with expression and return type */

val x = when{
price <= 19 -> "Free"
price <= 29 -> "Perfect price"
else -> "Exceptional price"
}
Perfect price

x
kotlin.Unit

==================ARRAY====================
val array = arrayOf(2,3,4,5,7)

array
[Ljava.lang.Integer;@1a2cbe42
```

```
array.joinToString()
2, 3, 4, 5, 7

val array = intArrayOf(2,3,4,5,7)
val list = listOf(1,1,2,3,5,8,13)

list[0] = 9
error: unresolved reference. None of the following candidates is
applicable because of receiver type mismatch:
@InlineOnly public operator inline fun <K, V> MutableMap<Int,
Int>.set(key: Int, value: Int): Unit defined in kotlin.collections
@InlineOnly public operator inline fun kotlin.text.StringBuilder /* =
java.lang.StringBuilder */.set(index: Int, value: Char): Unit defined
in kotlin.text

val mutableList = mutableListOf(1,1,2,3,5,8,13)
mutableList[0] = 99

mutableList
[99, 1, 2, 3, 5, 8, 13]

val set = setOf(1,1,2,4,5)

set
[1, 2, 4, 5]

val mutableSet = mutableSetOf(1,1,2,4,5)

mutableSet
[1, 2, 4, 5]

val map = mapOf(Pair(1, "Kotlin"), Pair(2, "Android"))

map
{1=Kotlin, 2=Android}

// Optional declaration
val map: Map<Int, String> = mapOf(Pair(1,"Kotlin"), Pair(2,
```

```
"Android"))

map
{1=Kotlin, 2=Android}

val mutableMap = mutableMapOf(1 to "Java", 2 to "Dart")

mutableMap
{1=Java, 2=Dart}

// Shallow copy
val list = set.toList()

list
[1, 2, 4, 5]

==========FOR LOOP ==================
for (i in 1..10){
println("$i ")
}
1 2 3 4 5 6 7 8 9 10

for( c in "Kotlin"){
print("$c ")
}
K o t l i n

for (i in 1..10 step 2){
println("$i ")
}
1 3 5 7 9

for (i in 10 downTo 1){
println("$i ")
}
10 9 8 7 6 5 4 3 2 1

val languages = listOf("Kotlin", "Java", "Swift")
```

```
for (lang in languages){
println("$lang is great")
}
Kotlin is greatJava is greatSwift is great

==========WHILE LOOP ==========
while(i <= 10){
println("$i ")
i++
}
==========FUNCTIONS==========
fun permitEntrance(age : Int) : Boolean {
return age >= 18
}

permitEntrance(7)
false

permitEntrance(18)
true

fun permitEntrance(age : Int) : Boolean = age >= 18

permitEntrance(20)
true

fun permitEntrance(vararg ages: Int) : Boolean {
return ages.any { age -> age >= 18}
}

permitEntrance(7, 2, 17)
false

permitEntrance(7, 2, 18)
true

==========USER INPUT==========
package basics
```

```
val PI = 3.1415
fun main(args : Array<String>){
var user: String?= "anonymous"
user = readLine()

val name = if(user!=null && user.isNotBlank())
println("Greetings! $user")
else
println("Greetings unknown")
}

===== COLLECTION LOOP EXAMPLE ==
package basics

import java.util.Random

fun main(args : Array<String>){

val array = mutableListOf<Int>()

for (i in 1..100){
array.add(Random().nextInt(100) + 1)
}

var i = 0
while(i < array.size && array[i] >= 10){
println(array[i])
i++
}
}
======PASSING FUNCTION PARAM ======

fun main(args : Array<String>){
var message = concat(separator = " : ", texts = listOf("Android",
"Java", "Kotlin"))
println(message)
}
```

```
fun concat(texts: List<String>, separator : String = ", ") =
texts.joinToString(separator)
=====LAMBDAS & HIGH ORDER FUNCTIONS====
fun main(args: Array<String>) {

val timesTwo = { x : Int -> x*2 }

val sumTwo: (Int, Int) -> Int = { x : Int, y : Int -> x +y }

val list = (1..100).toList()

println(list.filter ({ element -> element % 2 ==0 }))

println(list.filter( {it % 2 == 0}))

println(list.filter({it.even()}))

println(list.filter(::isEven))
}

fun isEven(i:Int) = i%2 ==0
fun Int.even() = this %2 ==0

====MAN & FLATMAP==========
fun main(args: Array<String>) {

// map()
val list = (1..100).toList()
val doubled = list.map({element -> element * 2})
val tripled = list.map({it * 3})
val average = list.average()
val shifted = list.map{it - average}
println(doubled)
println(tripled)
println(shifted)

// flatmap()
val nestedList = listOf(
```

```
(1..10).toList(),
(11..20).toList(),
(21..30).toList()
)

val notFlattened = nestedList.map{it.sortedDescending()} //.flatten()
val flattened = nestedList.flatMap{it.sortedDescending()}

println(notFlattened)
println(flattened)
}

======TAKE & DROP===============
fun main(args: Array<String>) {

val list = (1..1000).toList()

val first10 = list.take(10)
val last100 = list.drop(900)

//println(first10)
//println(last100)

val list1 = generateSequence(0) {
println("calculating... ${it+10}")
it + 10
}

val first10s = list1.take(10).toList()
val first20s = list1.take(20).toList()

println(first10s)
println(first20s)
}

=====ZIP =================
fun main(args: Array<String>) {
```

```
val list = listOf("Hi", "Kotlin", "there", "fans")
val containsT = listOf(false, true, true, false)

val zipf : List<Pair<String, Boolean>> = list.zip(containsT)
val mapping = list.zip( list.map{ it.contains("t")})

println(zipf)
println(mapping)
}

==== UNZIP & Example =========
fun main(args: Array<String>) {

val data = mapOf(
"users1.csv" to listOf(32, 45, 17, -1, 34),
"users2.csv" to listOf(19, -1, 67, 22),
"users3.csv" to listOf(),
"users4.csv" to listOf(56, 32, 18, 44)
)

val flattenedData = data.flatMap{it.value} //.filter { it > 0 }
val validAge = flattenedData.filter { it > 0 }
val averageAge = validAge.average()

val inValidAge = flattenedData.filter { it <= 0 }

val fileName = (data.toList().filter { it.second.toList().any{ num ->
num <= 0}}).unzip()

val filename = data.filter { it.value.any{ it < 0} }.map { it.key }
println(averageAge)
println(fileName.first)
println(inValidAge.count())
}

====CHAINING EXAMPLE==============
fun main(args: Array<String>) {
```

```
val inputRows = listOf(
mapOf("customfile1.csv" to listOf(3,4,5,6 -993, 53, 30, 10)),
mapOf("customfile2.csv" to listOf(13,14,5,6, 9930, 153, 30, 10)),
mapOf("customfile3.csv" to listOf(32,24,5,6, 93, 513, 30, 10)),
mapOf("customfile4.csv" to listOf(35,34,5,6 -193, 531, 310, 310))
)

val cleaned = inputRows.flatMap { it.values }
.flatten()
.filter { it>0 }
.toIntArray()

println(cleaned.joinToString())
}

======LAZY SEQUENCE==========

fun main(args: Array<String>) {

val verylonglist = (1..9999999L).toList()

val sum = verylonglist
.asSequence()
.filter{it>50}
.map{it*2}
.take(1000)
.sum()

val seq = generateSequence(1, {it + 1})

println(sum)
println(seq.take(10).toList())
}

========Working with Nullables ========
import java.io.File

fun main(args: Array<String>) {
```

```
//let()

// Scoping
File("example.txt").bufferedReader().let {
if(it.ready()){
println(it.readLine())
}
}

// Working with nullables
val str: String? = "Kotlin is fun"

str?.let{
if(str.isNotEmpty())
println(str.toLowerCase())
}
}

========WITH function=====
fun main(args: Array<String>) {

val props = System.getProperties()

with(props){
list(System.out)
println(propertyNames().toList())
println(getProperty("user.name"))
}
}

====USE keyword====
import java.io.FileReader

fun main(args: Array<String>) {

FileReader("example.txt").use{
val str = it.readText()
println(str)
```

```
}

}

=====INLINE function====

//Just runs the function body in the same line
inline fun modifyStr(str : String, operation : (String) -> (String)) :
String {
return operation(str)
}

fun main(args: Array<String>) {
println(modifyStr("Kotling is Amazing", { it.toLowerCase()}))
}

=====Classes & Properties===

class City{
var name:String = ""
var population:Int = 0
}

fun main(args: Array<String>) {

val berlin = City()
berlin.name = "berlin"
berlin.population = 3_50_0000

println(berlin.name)
println(berlin.population)
}

====PRIMARY and SECONDARY CONSTRUCTORS===

class City{

var name: String = ""
```

```
var population: Int = 0
}
class Country(val name: String, val areaSqKm: Int){

constructor(name : String) : this(name, 0){
println("Constructor called")
}
fun print() = "$name, $areaSqKm km^"
}
fun main(args: Array<String>) {

val bangladesh = Country("bangladesh", 157000)
println(bangladesh.name)
println(bangladesh.areaSqKm)

val australia = Country("australia")
println(australia.name)
println(australia.areaSqKm)
}

===METHODS in a CLASS========
class Robot(val name: String){

fun greetHuman(){
print("Hello Human, my name is $name")
}

fun knowsItsNam() : Boolean = name.isNotBlank()
}

fun main(args: Array<String>) {

val myRobot = Robot("tesla9000")

if(myRobot.knowsItsNam())
myRobot.greetHuman()
}
```

```
==== Extension Function==========

fun Int.isEven() = (this % 2 ==0)

fun City.isLarge() = population > 1_00_0000

fun main(args: Array<String>) {
println(5.isEven())

val naturals = listOf(2,5,8,13,21,22)
println(naturals.filter { it.isEven() })

val austin = City()
austin.name = "Austin"
austin.population = 90000
println(austin.isLarge())
}

====DATA Classes ==============
data class Address(val street: String, val number : Int, val postCode:
String, val city:String)
fun main(args: Array<String>) {

val res1 = Address("Main street", 42, "1234", "New York")
val res2 = Address("Main street", 42, "1234", "New York")

println(res1)
// Referential Equality
println(res1 === res2)
// Structural Equality
println(res1 == res2)

// copy
val neighbour = res1.copy(number = 43)
println(neighbour)

// Destructuring operation
res1.component1()
```

```
val (street, number, postCode, city) = res1
println("$street $number $postCode $city")
}

====ENUM================
enum class Direction(degree: Double){
NORTH(0.0), EAST(90.0), SOUTH(180.0), WEST(270.0)
}

enum class Suits{
SPADES, HEARTS, DIAMONDS, CLUBS
}

fun main(args: Array<String>) {

val suit = Suits.SPADES

val color = when(suit){
Suits.SPADES, Suits.CLUBS -> "red"
Suits.HEARTS, Suits.DIAMONDS -> "black"
}

println(color)
}

========INHERITENCE============
open class Shape(val name: String){
open fun area() = 0.0
}

class Circle(name: String, val radius : Double) : Shape(name){
override fun area() = Math.PI * Math.pow(radius, 2.0)
}

fun main(args: Array<String>) {
val smallCircle = Circle("Small Circle", 2.0)

println(smallCircle.name)
```

```
println(smallCircle.radius)
println(smallCircle.area())
}
```