Kotlin Cheat Sheet

Declaring Constants

```
// const can only be used in top
level or objects
const val PI = 3.14

// otherwise, use val for constant
val URL =
  "calendar.learn2develop.net"
```

Declaring Variables

```
val num = 5       // type inference
val num2:Int = 6  // explicit type
val msg = "Welcome!"
val name:String
```

Type Alias

```
typealias CustomerIDType   = Int
typealias CustomerNameType = String

var customerID: CustomerIDType
var customerName: CustomerNameType

customerID = 12345
customerName = "Samanta Choo"
```

Tuples (Pair/Triple)

```
var pt = Pair(7,8)
var flight =
  Triple(7031, "ATL", "ORD")

var (flightno, orig, dest) = flight

print(flightno)  //---7031---
print(orig)      //---ATL---
print(dest)      //---ORD---

print(flight.first)  //---7031---
print(flight.second) //---ATL---
print(flight.third)  //---ORD---
```

Nulls

```
var insurance_no:String? =
  "12345678"
insurance_no = null
```

Checking for Null

```
if (insurance_no != null) {
  val length = insurance_no.length
}
```

Safe Call Operator - ?

```
insurance_no = null
val length = insurance_no?.length
  // length is null

val fruits:List<String?> =
  listOf("apple", "orange", null)

for (fruit in fruits) {
  fruit?.let {
    print(fruit)
  }
}
```

Non-null Assertion Operator - !!

```
var insurance_no:String? =
  "12345678"
val length = insurance_no!!.length
  // length is 8
```

Elvis Operator - ?:

```
insurance_no = null

val length =
  insurance_no?.length ?: -1
  // length is -1

var gender:String? = null
var genderOfCustomer =
  gender ?: "male" // male

gender = "female"
genderOfCustomer =
  gender ?: "male" // female
```

Casting

```
val num1 = 5     // Int
val num2 = 1.6   // Double
val num3 = 3.4F  // Float
val num4 = num1.toFloat()  // 5.0
val num5 = num2.toInt()    // 1
val str = num3.toString()  // "3.4"
val num6 = str.toInt() // exception
```

Using let with Null

```
fun getProductCode(code:String)
:String? {
  val code =
    if (code=="Diet Coke") "12345"
else null
  return code
}

val code =
  (getProductCode("Diet Coke"))
code?.let {
    print(code)
} ?:run {
    print("Code not found")
}
```

Strings

```
var str1 = "A string"
var str2:String = "A string"
var str3 = str1 + str2
var str4 = "A" + " " + "String"
```

String Interpolation

```
var firstName = "Wei-Meng"
var lastName:String = "Lee"

var fullName =
  "$firstName $lastName"
  // "Wei-Meng Lee"

var s =
"Value of \$firstName is $firstName"
// "Value of $firstName is Wei-Meng"
```

Unicode

```
val hand = '\u270B'
val star = '\u2b50'
```

Enumerations

```
enum class BagColor(val v:Int) {
  Black(1),
  White(2),
  Red(3),
  Green(4),
  Yellow(5),
}

enum class DayOfWeek(
  val short:String, val long:String)
{
  Sunday ("Sun","SUNDAY"),
  Monday ("Mon","MONDAY"),
  Tuesday ("Tue","TUESDAY"),
  Wednesday ("Wed","WEDNESDAY"),
  Thursday ("Thu","THURSDAY"),
  Friday ("Fri","FRIDAY"),
  Saturday ("Sat","SATURDAY"),
}

var colorOfBag:BagColor
colorOfBag = BagColor.Red

var colorStr:String =
  colorOfBag.toString()
  // colorStr is now "Red"

val color = colorOfBag.v
  // color is 3

colorOfBag =
  BagColor.valueOf("Yellow")
// colorOfBag is now BagColor.Yellow

var day:DayOfWeek
day = DayOfWeek.Friday
val s1 = day.long  // "FRIDAY"
val s2 = day.short // "Fri"

day = DayOfWeek.valueOf("Saturday")
// day is now DayOfWeek.Saturday
```

Characters

```
// String
var euroStr = "€"

// Character
var euro:Char = '€'

// €2500
var price = euro + "2500"
// price is now a String
```

Range Operators

```
// Closed Range Operator
// prints 5 to 9 inclusive
for (num in 5..9 step 1) {
  print(num)
}

//---prints 9 to 5 inclusive---
for (num in 9 downTo 5 step 1) {
  print(num)
}

//---Half-Open Range Operator
//---prints 5 to 8---
for (num in 5 until 9 step 1) {
  print(num)
}
```

Functions

```
fun addNums(num1:Int,
            num2:Int,
            num3:Int):Int {
  return num1+num2+num3
}

val sum = addNums(5,6,7)
```

Returning Tuple

```
fun countNumbers(string:String):
Pair<Int,Int>{
  var odd = 3
  var even = 4
  // do something
  return Pair(odd, even)
}

val count = countNumbers("12345")
```

Default Parameter Value

```
fun join(firstName:String,
         lastName:String,
         joiner:String=" "):String {
  return $firstName$joiner$lastName"
}

var fullName =
  join("Wei-Meng", "Lee", ",")
  // Wei-Meng,Lee

fullName = join("Wei-Meng", "Lee")
  // Wei-Meng Lee
```

Variable Arguments

```
fun average(vararg nums:Int):Float {
  var sum: Float = 0F
  for (num in nums) {
    sum += num.toFloat()
  }
  return sum/nums.size
}

var avg = average(1,2,3,4,5,6)
  // 3.5
```

Arrays and Lists

```
var names = Array<String>(5) {""}
// 5 elements all containing ""

var OSes: Array<String> =
  arrayOf("iOS",
          "Android",
          "Windows Phone")

var os1 = OSes[0] // "iOS"
var os2 = OSes[1] // "Android"
var os3 = OSes[2] // "Windows Phone"
var count = OSes.size // 3

// Array is mutable
OSes[2] = "Fuchsia"

var items: List<Int> =
  listOf(1,2,3,4,5)
var item1 = items[0] // 1
var item2 = items[1] // 2
var item3 = items[2] // 3

// error
// items[3] = 9 // list is immutable

// mutable (dynamic)
var wishes: MutableList<String> =
  mutableListOf()
wishes.add("iPhone")
wishes.add("iPad")
```

Array<Int> vs IntArray

```
// declare; numbers0 is now null
val numbers0: Array<Int>
// same as Integer[] in Java

// declare and init; numbers1 now
// has 10 elements
var numbers1: Array<Int> =
    arrayOf(0,1,2,3,4,5,6,7,8,9)

// same as numbers1
var numbers2=
    arrayOf(0,1,2,3,4,5,6,7,8,9)

//---numbers3 has now 10 elements
// all 0s
var numbers3 = IntArray(10)
// same as int[] in Java

//---numbers3 has now 10 elements
// from 0 to 9
numbers3 =
  intArrayOf(0,1,2,3,4,5,6,7,8,9)

// numbers4 has now 10 elements
// {0,1,4,9,16,25,...}
val numbers4 = IntArray(10) {
  n -> n*n
}
```

The it Keyword

```
// same as numbers4
val numbers5 = IntArray(10) {
    it*it
   // "it" is the implicit
   // name of the single parameter
}
```

Dictionaries

```
val platforms1 = hashMapOf(
  "Apple" to "iOS",
  "Google" to "Android",
  "Microsoft" to "Windows Phone"
)

val p1 = platforms1["Apple"]
//---"iOS"---

val p2 = platforms1["Samsung"]
//---null---

val count = platforms1.size
val companies = platforms1.keys
val oses = platforms1.values

platforms1["Samsung"] = "Bada"
```

```
val months:HashMap<Int, String> =
  hashMapOf()
months[1] = "Jan"
months[2] = "Feb"
```

The when Statement (Switch)

```
var grade: Char = 'B'
when (grade) {
  'A', 'B', 'C', 'D' ->
    print("Passed")
  'F' ->
    print("Failed")
  else ->
    print("Undefined")
}
```

Matching Range

```
var percentage: Int = 85
when (percentage) {
  in 0..20 ->
    print("Group 1")
  in 21..40 ->
    print("Group 2")
  in 41..60 ->
    print("Group 3")
  in 61..80 ->
    print("Group 4")
  in 81..100 ->
    print("Group 5")
  else ->
    print("Invalid percentage")
}
```

Data Class

```
data class Go (
  var row:Int,
  var column:Int
)
var stone1 = Go(12,16)
with(stone1) {
  row = 9
  column = 13
}
```

Looping

```
// prints 0 to 4
var count = 0
while (count < 5) {
  print (count)
  count += 1
}

// prints 0 to 4
count = 0
repeat (5) {
  print(count)
  count += 1
}

// prints 0 to 4
for (i in 0..4) {
    print(i)
}
```

Classes

```
class MyLocation {
}

// type inference
val loc1 = MyLocation()

// declare and initialize
val loc2:MyLocation = MyLocation()
```

Late Initialization

```
// late initialization (only applies
// to var)
lateinit var loc3:MyLocation
...
loc3 = MyLocation()
```

Lazy Initialization

```
// lazy only applies to val
public class Example{
  // defer initialization of an
  // object until the point at
  // which it is needed.
  val name: String by lazy {
    "Johnson"
  }
}

val e = Example()
val name = e.name  // "johnson"
```

Properties

```
class MyLocation {
  // all properties must be
  // initialized
  // var for read/write properties
  var lat:Double? = null
  var lng:Double? = null

  // val for read-only properties
  val arrived:Boolean = false
}

val loc = MyLocation()
loc.lat = 57.474392
loc.lng = 37.228008
print(loc.arrived)
```

Primary Constructor

```
// primary constructor
class MyLocation (var lat:Double,
                  var lng:Double) {
  val arrived:Boolean = false
```

```
  fun someMethod() {
    val latitude = this.lat
    val longitude = this.lng
  }
}

var loc = MyLocation(57.474392,
                     37.228008)
```

Primary Constructor with Initializer Block

```
class MyLocation (var lat:Double,
                  var lng:Double) {
  val arrived:Boolean = false

  init {
    this.lat = lat
    this.lng = lng
    // you can add more code here
  }

  fun someMethod() {
    val latitude = this.lat
    val longitude = this.lng
  }
}
```

Secondary Constructor

```
class MyLocation {
  var lat:Double
  var lng:Double
  var arrived:Boolean = false

  // secondary constructor
  constructor(lat:Double,
              lng:Double) {
    this.lat = lat
    this.lng = lng
    // you can add code here
  }

  // secondary constructor; chaining
  constructor(lat:Double,
              lng:Double,
              arrived:Boolean):
              this(lat,lng) {
    this.arrived = arrived
  }
}

var loc = MyLocation(57.474392,
    37.228008)
var loc2 = MyLocation(57.474392,
    37.228008, true)
```

Custom Getter and Setter

```
// a.k.a. computed properties
class MyLocation () {
  var lat:Double = 0.0
  set(value) {
    if (value > 90 ||
      value < -90) {
      throw
        IllegalArgumentException(
        "Invalid latitude")
    }
    field = value
  }
  var lng:Double = 0.0
  set(value) {
    if (value > 180 ||
        value < -180) {
      throw
        IllegalArgumentException(
        "Invalid longitude")
    }
    field = value
  }
  val arrived:Boolean
  get() {
    return (this.lat in 57.0..58.0
         && this.lng in 37.0..38.0)
  }
}

var loc5 = MyLocation()
loc5.lat = 157.474392 // exception
loc5.lng = 37.228008
print(loc5.arrived)
```

```
class Distance {
  var miles = 0.0
  var km: Double
  set (km) {
    miles = km / 1.60934
  }
  get() {
    return 1.60934 * miles
  }
}

var d = Distance()
d.miles  = 10.0
print(d.km) // 16.0934
```

Class Inheritance

```
// use open to allow the class to be
// inherited
open class BaseClass(arg1:String) {
  // class has a primary
  // constructor with 1 argument
}

// initializing the parent class via
// the class header
class SubClass1(
  arg1:String,
  arg2:String):BaseClass(arg1) {
}

// initializing the parent class via
// the secondary constructor using
// the super keyword
class SubClass2:BaseClass {
  constructor(arg1:String,
              arg2:String):
              super(arg1) {
  }

  // constructor chaining
  constructor(arg1:String):
    this(arg1, "") {
  }
}
```

```
open class Shape(length:Float,
                 width:Float) {
  var length:Float
  var width:Float

  init {
    this.length = length
    this.width = width
  }
  fun perimeter():Float {
    return 2 * (length + width)
  }
  fun area(): Float {
    return length * width
  }
}

open class Rectangle(length:Float,
                     width:Float):
  Shape(length,width) {
  private val INCH_TO_CM = 2.54F
  init {
    this.length = length *
                  INCH_TO_CM
    this.width = width * INCH_TO_CM
  }
}

class Square(length:Float):
  Rectangle(length, length) {
}
```

Lambda Functions

```
//---Button view---
val btnOpen =
  findViewById<Button>(R.id.btnOpen)

// traditional way
btnOpen.setOnClickListener(object:
  View.OnClickListener{
  override fun onClick(v: View?) {
    ...
  }}
)
// alternative 1
btnOpen.setOnClickListener(
  { view ->
    ...
  })
// alternative 2
btnOpen.setOnClickListener()
  { view ->
    ...
  }
// alternative 3
btnOpen.setOnClickListener {
  view ->
    ...
}
// alternative 4
btnOpen.setOnClickListener {
  ...
}
```

```
var numbers =
  arrayOf(5,2,8,7,9,4,3,1)

// return all even numbers
val evenNumbers = numbers.filter {
  n -> n % 2 == 0
}

// sum up all the numbers
val sums = numbers.reduce {
  sum, n -> sum + n
}

// convert the list to strings,
prefix each item with "$"
val prices = numbers.map{
  n -> "\$$n"
}

// apply GST to items above 3,
// convert the list to strings,
// prefix with "$"
val pricesWithGST = numbers.map{
 n -> "$" + if (n>3) (n*1.07) else n
}

// sort in ascending order
val sortedASCNumbers =
numbers.sortedBy {
  it
}

// sort in descending order
val sortedDESCNumbers =
numbers.sortedBy {
  it
}.reversed()
```

Accepting Lambda Functions as Arguments

```
fun bubbleSort(
  items:IntArray,
  compareFun:(Int, Int) -> Boolean)
  :IntArray {

  for (j in 0..items.size-2) {
    var swapped = false
    for (i in 0..items.size-2-j) {
      // you need to swap if the
      // numbers are not in order
      if (!compareFun(items[i],
```

```
                    items[i+1])) {
        val temp = items[i+1]
        items[i+1] = items[i]
        items[i] = temp
        swapped = true
      }
    }
    if (!swapped) break
  }
  return items
}

val numbers =
  intArrayOf(5,2,8,7,9,4,3,1)

// ascending order
var sortedNumbers = bubbleSort(
    numbers, { num1, num2 ->
      num1 < num2 } )

// descending order
sortedNumbers = bubbleSort(
    numbers, { num1, num2 ->
      num1 > num2 } )

// another way of passing in the
// lambda function
sortedNumbers = bubbleSort(numbers)
  { num1, num2 -> num1 < num2 }
```

Property Observers

```
class MyPointClass {
  var x: Int by
    Delegates.observable(0) {
     prop, old:Int, new:Int ->
       println(prop.name +
        " : from $old to $new")
    }
}

val pt = MyPointClass()
pt.x = 5 // x : from 0 to 5
pt.x = 6 // x : from 5 to 6
```

Vetoable Property

```
class MyPointClass {
  var x: Int by
    Delegates.vetoable(0) {
     prop, old:Int, new:Int ->
       println(prop.name +
         " : from $old to $new")
       new > 0
    }
}

val pt = MyPointClass()
pt.x = 5 // x : from 0 to 5
pt.x = 6 // x : from 5 to 6
pt.x = 0 // x will roll back to 6
```

Identity Operator

```
class MyPointClass {
  var x = 0
  var y = 0

  constructor(x:Int, y:Int) {
    this.x = x
    this.y = y
  }
}
var pt1 = MyPointClass(5,6)
var pt2 = pt1
var pt3 = MyPointClass(5,6)
if (pt1 === pt2) {
  print("Identical")
} else {
  print("Not identical")
} // Identical
if (pt1 === pt3) {
  print("Identical")
} else {
  print("Not identical")
} // Not identical
```

Instance Methods

```
class Car {
  var speed = 0
  fun accelerate() {
  }
  fun decelerate() {
  }
  fun stop() {
  }
  fun printSpeed() {
  }
}

val c = Car()
c.accelerate()
c.decelerate()
c.stop()
c.printSpeed()
```

Companion Object (Static Method/Variable)

```
class Car {
  companion object {
    val MilesToKM = 1.60934
    fun kilometersToMiles(
      km:Float):Double {
        return km / 1.60934
    }
  }
  fun accelerate() {}
  fun decelerate() {}
  fun stop() {}
  fun printSpeed() {}
}
val v = Car.MilesToKM
val miles =
Car.kilometersToMiles(10F)
```

Final Class

```
// by default, all the classes in
// Kotlin are final (non-
// inheritable)
class ClosedClass {
}
// error
class Subclass0:ClosedClass() {
}
```

Overriding and Overloading Methods

```
open class Shape(length:Float,
                 width:Float) {
  var length:Float
  var width:Float
  init {
    this.length = length
    this.width = width
  }
  open fun perimeter():Float {
    return 2 * (length + width)
  }
  open fun area(): Float {
    return length * width
  }
}

class Circle(diameter:Float):
  Shape(diameter / 2, diameter / 2)
{
  override fun area():Float {
    return (Math.PI *
      this.length.pow(2F)).toFloat()
  }

  override fun perimeter(): Float {
    return (2 * Math.PI *
      this.length).toFloat()
  }

  // overloading
  fun perimeter(radius:Float): Float
  {
    return (2 * Math.PI *
      radius).toFloat()
  }
}

var c = Circle(6F)
area = c.area() // 28.274334
perimeter = c.perimeter()
  // 18.849556
perimeter = c.perimeter(5F)
  // 31.415926
```

Extensions

```
// extending the Context class with
// the displayToast() function
fun Context.displayToast(
  text:CharSequence,
  duration:Int=Toast.LENGTH_SHORT) {
    Toast.makeText(this, text,
        duration).show()
}
// without extension
Toast.makeText(this,
   "Hello, Kotlin",
   Toast.LENGTH_SHORT).show()
// with extension
displayToast("Hello, Kotlin!")
```

```
fun String.LatLng(splitter:String):
  Pair<Float, Float> {
  val latlng = this.split(splitter)
  return Pair(latlng[0].toFloat(),
              latlng[1].toFloat())
}
var str = "1.23456,103.345678"
var latlng  = str.LatLng(",")
print(latlng.first)
print(latlng.second)
```

Interfaces

```
interface CarInterface {
  fun accelerate() {
    // default implementation
  }
  fun decelerate()
  fun accelerateBy(amount:Int)
}

class MyCar:CarInterface {
  override fun accelerate() {
    // override implementation in
    // interface
  }
  override fun decelerate() {
  }
  override fun accelerateBy(amount:
    Int) {
  }
}
```

Generic Class

```
class MyStack<T> {
  val elements:MutableList<T> =
    mutableListOf()
  fun push(item:T) {
    elements.add(item)
  }
  fun pop():T? {
    if (elements.size>0) {
      return elements.removeAt(
        elements.size-1)
    } else {
      return null
    }
  }
}
var item = myStringStack.pop()
  // kotlin
item = myStringStack.pop()
  // programming
item = myStringStack.pop() // null
```