Basic Kotlin Cheat Sheet

Comment blocks: C++/Java style `//` and `/* */`
Variables: `var` = mutable, `val` = immutable, `?` nullable. Example `var myString : String? = null`
Types: `Byte`, `Short`, `Int`, `Long`, `Float`, `Double`, `Char`, `Boolean` (`true|false`), `String`, `Any` (Object or primative). Numbers can have underscores to help readability. `1_000_000, 0xFF_CB, 0b10011001_10101010`
OOP Keywords: `class`, `object`(singleton), `interface`. **Modifiers**: `open`, `final`, `const`, `companion`, `abstract`, `sealed`
Operators – Maths:`+` (`plus`), `-` (`minus`), `/` (`div`), `%` (`mod`), `*` (`times`) , `+=` (`plusAssign`), `-=` (`minusAssign`), etc… `+a` (`unaryPlus`), `-a` (`unaryMinus`), `!a` (`not`), `++a`, `a++` (`inc`), --a, a—(dec), `==`, != (`equals`), <, > , >=, <= (`compareTo`)
`===` compares references. `(null === null) == true`.
Conditionals
`if` is a function. So `var myInt = if (true) then 42 else 0` will assign 42 to myInt. No :? Operator
`when` replaces switch/case. You can constants and arbitrary expressions. Execution stops when condition satisfied.

```
when (x) {
      0,1 -> print(“0 or 1”)
      parseInt(s) -> print(“s encodes x”)
      in 10..100->print(“between 10 and 100”)
      else -> print(“something else”)
}
```

when can be used to replace if/else/if chains

```
when {
      x<0 -> print(“x is minus”)
      x in 1..10 -> print(“x is in range”)
      else -> print(“out of range”)
}
```

Type casting and conversion: is, as, as?
`if (myObject is String) print(x.length);` `x` is smart cast to `String`
`as` considered unsafe. Use `as?`. `Val x: String? = y as? String`
Exceptions: `try catch finally`. Try can be used as an expression:

```
val a: Int? = try { parseInt(input) } catch (e: NumberFormatException) { null }
```

Flow: `while`/`do while` same as Java/C++
`for` is the equivalent to `foreach` in other languages.

```
for (item in collection) print(item)
for (counter in lowerBound..upperBound) print(counter)
```

`break`/`continue` is supported in loops
Classes: Classes and members are public and final by default. Mark classes with `open` if required.
Trivial Constructor
`class Person(val name: String, var age: Int)` – creates name readonly field and age mutable field.

Non trivial Constructor

```
class Person(name: String) {
      var firstName : String
      var secondName: String
      init {
            val separate = name.split(“ “)
            firstName = separate[0]
            secondName = separate[1]
      }
}
```

Multiple constructors with super classes. Shows calling super and delegated constructors.

```
class MyView : View {
    constructor(ctx: Context) : this(ctx, MY_STYLE) //Delegated constructor
    constructor(ctx: Context, attrs: AttributeSet) : super(ctx, attrs)
}
```