# Coroutines Cheat Sheet

**Coroutines**: Handle long running tasks elegant and efficiently (**Asynchronous** – **Non-blocking** – **Sequential code**).

**Suspend:** Function type that suspend thread execution until the result is ready. While it's suspended, it unblock the threads that it's running on. So other functions or coroutine can run.

Note: suspended function can run on main thread or background thread.

## How to implement it?

1. **Job:** Background Job.
2. **Dispatcher:** Determines the thread.
3. **Scope:** Combine information including a job and dispatcher, to define the context in which coroutine run.

## Scope Types

1. **GlobalScope** - Lifetime of the new coroutine is limited only by the lifetime of the whole application
2. **CoroutineScope** - Is destroyed after all launched children are completed
3. **MainScope** - Scope for UI applications and uses Dispatchers.Main

Example

```
private val job = Job()
private val coroutineScope =
CoroutineScope(job + Dispatchers.Main)

fun someWorkNeedToBeDone() {
   coroutineScope.launch {
      suspendedFun()
  }
}
//suspended fun start another coroutine scope
suspend fun suspendedFun() {
   withContext(Dispatchers.IO) {
      //Long Running Task & it can return value
   }
}
```

## Async

It is like launch {}, The difference is that launch returns a Job and does not carry any resulting value, while async returns a **Deferred**, which has an **await()** function that returns the result of the coroutine.

```
fun deferredFunAsync(): Deferred<Int> {
   return coroutineScope.async {
      return@async 0
   }
}
fun someWorkNeedToBeDone() {
   coroutineScope.launch {
      try {
         val value = deferredFunAsync().await()
      } catch (t: Throwable) { }
   }
}
```

## Blocking

**Run Blocking** runs a new coroutine and blocks the current thread, interruptible until its completion

```
runBlocking {
   //Delay is non-blocking
   delay(2000)

}
```

## Channel

It could be considered to provide a stream of values between coroutines. It is very similar to BlockingQueue.

```
val channel = Channel<Int>()
// Coroutines#1
MainScope().launch
{
   (1..5).forEach {
      channel.send(it) }

}

// Coroutines#2
MainScope().launch
{
   repeat(5) {
      Log.d("demo", "${channel.receive()}")
   }
   Log.d("demo", "done")
}
```