协程，又称微线程，纤程。英文名Coroutine。

协程的概念很早就提出来了，但直到最近几年才在某些语言（如Lua）中得到广泛应用。

子程序，或者称为函数，在所有语言中都是层级调用，比如A调用B，B在执行过程中又调用了C，C执行完毕返回，B执行完毕返回，最后是A执行完毕。

所以子程序调用是通过栈实现的，一个线程就是执行一个子程序。

子程序调用总是一个入口，一次返回，调用顺序是明确的。而协程的调用和子程序不同。

协程看上去也是子程序，但执行过程中，在子程序内部可中断，然后转而执行别的子程序，在适当的时候再返回来接着执行。

注意，在一个子程序中中断，去执行其他子程序，不是函数调用，有点类似CPU的中断。比如子程序A、B：

def A():

print '1'

print '2'

print '3'

def B():

print 'x'

print 'y'

print 'z'

假设由协程执行，**在执行A的过程中，可以随时中断，去执行B，B也可能在执行过程中中断再去执行A**，结果可能是：

1

2

x

y

3

z

但是在A中是没有调用B的，所以协程的调用比函数调用理解起来要难一些。

看起来A、B的执行有点像多线程，**但协程的特点在于是一个线程执行**，那和多线程比，协程有何优势？

最大的优势就是协程极高的执行效率。因为**子程序切换不是线程切换**，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。

第二大优势就是不需要多线程的锁机制，因为只有一个线程，也**不存在同时写变量**冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多。

因为协程是一个线程执行，那怎么利用多核CPU呢？最简单的方法是多进程+协程，既充分利用多核，又充分发挥协程的高效率，可获得极高的性能。

进程需要自己拥有调度线程的能力。如果一种实现使得每个线程需要自己通过调用某个方法，主动交出控制权。那么我们就称这种用户态线程是协作式的，即是协程。