A better definition for the dilogarithm Li_2 is given by

$$\mathrm{Li}_2(z) := -\int_0^z dz' \, \frac{\log(1-z')}{z'} \tag{1}$$

This is well defined for arbitrary complex values z, with a branch cut along the real axis for $z > 1$. It can be shown that

$$\mathrm{Li}_2(1) = \frac{\pi^2}{6} \tag{2}$$

$$\mathrm{Li}_2(-1) = -\frac{\pi^2}{12} \tag{3}$$

We will try to establish that Eq. 2 and Eq. 3 are true by numerically evaluating the integral in Eq. 1 both sequentially and *in parallel* using threads and `MPI`.

1. Let's implement a function `dilog` that accepts a real number z and a potentially large integer N, and returns an approximation to Eq. 1. For example, a `C` or `C++` prototype would look like

```
double dilog(double z, long N);
```

   (a) First, take care of the obvious cases $n < 1$, $z = 0$ and, for the sake of simplicity, $z > 1$ where the dilogarithm picks up an imaginary part.

   (b) Implement the numerical approximation to the integral in Eq. 1 by splitting the interval $(0, z)$ in N subintervals of equal length dz. Each subinterval contributes an amount equal to its width multiplied by the value of the integrand at the center of the subinterval. Return the sum of these contributions as the approximation to the dilogarithm.

   (c) Why do we only evaluate the function at the center of each subinterval?

   (d) Write a driver for `dilog` and convince yourself that it gives reasonable approximations to the values in Eq. 2 and Eq. 3. Fill out the following table:

N	$\mathrm{Li}_2(-1)$	Δ	$\mathrm{Li}_2(1)$	Δ
10^1				
10^3				
10^6				
10^9				
10^{12}				

   where Δ is the relative error in each case.

2. Explain how you would compute an approximation of the integral in Eq. 1 *in parallel*. In particular:

   (a) What are the finest grain parallel tasks?

(b) What are the communication requirements among these tasks?

(c) What is the correct coarsening strategy?

(d) How you would map the coarse tasks onto physical processing units?

3. Implement a version of `dilog` that uses pthreads for enhanced performance.

   (a) What is the correct partitioning, coarsening and mapping strategies when using threads?

   (b) Implement the routine that becomes the body of each thread.

   (c) Implement `dilog` to perform the error checking, spawn n threads to perform the calculation, collect the results of each thread and return the approximation to the integral.

   (d) Convince yourself that `dilog` returns correct values regardless of the number of threads it spawned.

4. Implement a parallel version `dilog` using `MPI`.

   (a) What is the correct partitioning, coarsening and mapping strategies when using distributed memory parallelism?

   (b) Implement `dilog`, making sure that each processor has the correct return value from the function.

   (c) Convince yourself that `dilog` returns correct values regardless of the number of processors used.