SUMMARY OF MAIN FUNCTIONS

This folder contains the code which is more tailored towards the topics in [DV2] and has lots of additional routines. This is a summary of the most important ones.

To start, save the folder `gross_stark` on a machine that has a recent version of `magma`. The easiest thing to do is to log on to `elastigirl`, which will have that folder in the main directory already. Then make this folder your directory:

```
cd path-to-folder/gross_stark
magma
```

Once in magma, load the code by running:

```
> load "rmmc.m";
```

1. Cocycles

To compute a cocycle, simply run the command

```
> J := RigidCocycle(D321F2,7,100);
```

which will take about three minutes. Have a cup of coffee meanwhile. In principle, the command above can be used to compute "norms" of cocycles, even for ramified primes, or cocycles attached to split quadratic forms. However, for those there are still loose ends for me to tie up, so use those routines at your own risk. There is also a shorthand, if you prefer:

```
> J := rmc(D321F2,7,100);
```

The naming convention is still the same: All quadratic forms of discriminant less than a thousand are automatically initialised with the convention that `DaFb` stands for a form of discriminant `a` in the `b`-th equivalent class. In the above example, the cocycle of the non-trivial class of discriminant 12 was computed, to 5-adic precision 100.

Various operations on rigid cocycles can also be computed.[1] For instance, the following commands, with obvious variable names, compute powers, products, and quotients respectively

```
> Jn := CocyclePower(J,n);
> J1prodJ2 := CocycleProduct(J1,J2);
> J1divJ2 := CocycleDivision(J1,J2);
```

Also, Hecke operators are implemented. For instance, the following commands will compute the cocycles $(1 + w_p)J$, $(1 - w_p)J$, $(1 + w_\infty)J$, $(1 - w_\infty)J$, and $(T_l - a)J$ for $l \neq p$ and $a \in \mathbf{Z}$ respectively:

[1] Bear in mind that `magma` can be very formal and might refuse cooperation if universes are merely isomorphic and not identical, which I will try to bypass eventually.

```
> G := wpPlus(J);
> G := wpMinus(J);
> G := winfPlus(J);
> G := winfMinus(J);
> G := HeckeProjection(J,[l,a]);
```

Various other and related routines are available, but at this stage this will probably serve us quite well already.

2. RM values of cocycles

To obtain the RM value of a cocycle at the first root of a quadratic form, say `D5F1`, run

```
> a := Value(J,D5F1);
```

This will just give you the bare p-adic number, without any attempt to recognise it, since that attempt will depend on various features of the cocycle you are considering. For instance, the degree of the global number we are after may vary, as well as the ambiguity which could be roots of unity, or also powers of a fundamental unit. Depending on the situation, the following command for recognition are available:

```
> f := algdep(a,n);
> f, A := ThoroughAlgdepZeta(a,n);
> f, A := ThoroughAlgdepEpsilon(a,n,DF);
```

Here, `n` is the degree bound of the minimal polynomial we attempt to find, and `DF` is the discriminant of the order whose fundamental unit generates the ambiguity. The output is a guess for the minimal polynomial `f`, as well as the number `A` which is the same as `a` up to the ambiguity, and satisfies `f`. For instance, for the above example we would get:

```
> a := Value(wpPlus(J),D5F1);
> P, A := ThoroughAlgdepZeta(a,6);
> P;
75798639*x^6 - 133941654*x^5 + 101456577*x^4 - 62789236*x^3 + 101456577*x^2
- 133941654*x + 75798639
```

If one wants to check a large range of special values, and verify Shimura reciprocity at the same time, the following does most work automatically.

```
> G32 := RigidCocycle(D32F2,3,200);
> Values(wpPlus(G32),[1..100]);
```

If you feed it a cocycle whose ambiguity is only the torsion in $\mathbf{C}_p^\times$, which is the case for monstrous cocycles such as `wpPlus(J)`, you should add in the argument `sym := true` which causes tremendous speed-ups, but seems a bit buggy at times, I should look into this.

3. Gross–Stark units

Then, there are also routines to compute the quantities $\kappa_\tau[0,\infty]$. For instance, the following command returns this quantity for τ the first root of the form $[-4,15,6]$ of discriminant 321 which has class number 6.

```
> J := RigidCocycle(D321F2,7,100);
> a := GrossStark(J);
```

If one has enough precision and is lucky, this will work straight away, like it will with the parameters in this example. If the displayed polynomial looks suspicious, then it pays off to play around with '`a`', raise it to various powers, vary the degree of algebraic recognition routines, etc. If nothing works, increasing the precision should do.