Overview of the Project

This is the final project for the module 'Formalising Mathematics' and is based on the past examination paper of Number Theory of the year 2020 designed for Year 3/4 and M.Sc. I chose the first two parts of problem mentioned under Q4, which dealt with reflexive and corpulent integers. Thus, the aim of the project is as follows-
To show that :

1. An integer n is reflexive if and only if it is corpulent.
2. An integer n is corpulent if m is corpulent for every prime power $m|n$.

An integer $n \geq 1$ is **reflexive** if for all integers a, b with $(a, b) = 1$, the following two conditions are equivalent:

1. $a \equiv b \pmod{n}$.
2. $a * b \equiv 1 \pmod{n}$.

This is known as the diagonal property, which asks the question that when does Z_n have the property that whenever $a * b = 1$, $a = b$.

An integer $n \geq 1$ is **corpulent** if for all integers a with $(a, n) = 1$, we have $a^2 \equiv 1 \pmod{n}$. This is analogous to saying that if R is a ring then every unit of this ring is an involution.

In fact, this insight is provided in the paper arXiv:1212.3347 by Sunil K. Chebolu and Michael Mayers, which can be accessed via the URL: https://arxiv.org/pdf/1212.3347.pdf and gives a beautiful perspective to the first aim of the project!
To fulfill the aim of the project, I introduced several lemmas and theorems that I would outline in this report. I defined the two definitions under the name `reflexiv` and `corpulent` and then defined the two theorems that I am going to prove in this project as:

```
theorem reflexive_iff_corpulent : reflexiv hn ⟺ corpulent hn :=
```

This says that `hn`, which is defined as a hypothesis $(1 \leq n)$ for some integer n, is reflexive $\iff$ it is corpulent.

```
theorem corpulent_two_int: corpulent hn →   {p e : ℕ} {hp : nat.prime p}
{hm : (pow p e) | n} , corpulent (pow_pos (nat.pos_of_ne_zero
(nat.prime.ne_zero hp)) e) :=
```

This theorem defines that two integers are said to be corpulent if for $(1 \leq n)$ and $(1 \leq m)$, n is corpulent if m is corpulent for every prime power $m|n$, that is, for every divisor of n of the form p^e with $1 \leq e$.

The Mathematics and the Explanation of the Code

I started off by importing the necessary libraries and then proceeded to define the reflexive and corpulent integers under `reflexiv` and `corpulent`. The first definition for reflexive integers is not named as 'reflexive' because such a declaration in Lean already exists. The the definitions of the two terms in Lean, closely follow their mathematical structure and are defined as:

```
def reflexiv {n : ℕ} (hn : 1 ≤ n) :=
( {a b : ℤ} (h : a.gcd b = 1),  a ≡ b [ZMOD n]  ⟺  a * b ≡ 1 [ZMOD n])
```

```
def corpulent {n : ℕ} (hn : 1 ≤ n) :=
(  {a : ℤ} (han : a.gcd n = 1), a^2  ≡ 1 [ZMOD n])
```

My next step was to write the theorems that the project is based on and following their mathematical proofs, I one by one proceeded to write the main proof and create lemmas to help me prove it.

0.1 Mathematical Proof and Formalisation n is reflexive $\iff$ n is corpulent

Mathematical Proof of the forward direction:
To show : If n is reflexive, then it is corpulent :
n is reflexive $\rightarrow \exists a, b$ with $(a, b) = 1$, such that the following two conditions are equivalent:
$a \equiv b \pmod{n}$
$a * b \equiv 1 \pmod{n}$.
If we take $b = a + n$, then $(a, b) = 1$ as $(a, a + n) = (a, n) = 1$.
Clearly, putting the value of b in $a \equiv b \pmod{n}$ and $a * b \equiv 1 \pmod{n}$, gives, $a \equiv (a + n) \pmod{n}$ and $a * (a + n) \equiv 1 \pmod{n}$.
This implies that $a \equiv (a + n) \pmod{n} \implies a + 0 \equiv a + n \pmod{n}$,and
$a^2 + (a * n) \equiv 1 \pmod{n}$.
This gives that $a^2 \equiv 1 \pmod{n}$. Hence, n is a corpulent integer.

Formalising the proof of the forward direction:

```
  -- We will first prove the forward direction, i.e., `n` is reflexive →
  --`n` is corpulent.
    { unfold reflexiv,
      unfold corpulent,
      intros hp q hqn,
      -- The goal now is to show that `q ^ 2 ≡ 1 [ZMOD ↑n]`, given `n`, `hn`,
      --`hp`, `q`, `hqn`.
      have hp1 : q.gcd (q+n) = 1 → (q ≡ (q+n) [ZMOD ↑n]  ⟺  q * (q+n) ≡ 1 [ZMOD ↑n]),
```

```
  { exact hp},
have hp2 : (q ≡ q + ↑n [ZMOD ↑n]  ⟺  q * (q + ↑n) ≡ 1 [ZMOD ↑n]),
  { rw hp1,
    simp,
    rw hqn },
have hp3 : q * (q + ↑n) ≡ 1 [ZMOD ↑n],
  { rw ← hp2,
    apply int.modeq.symm,
    have h1 : q ≡ q [ZMOD ↑n] := int.modeq.rfl,
    have h2 : ↑n ≡ 0 [ZMOD ↑n],
    { rw int.modeq_zero_iff_dvd,},
    conv
      begin
        to_rhs,
        congr,
        skip,
        skip,
        rw ← add_zero q,
      end,
    apply int.modeq.add,
    exact h1,
    exact h2, },
have hp4: (q * ↑n) ≡ 0 [ZMOD ↑n],
{ rw int.modeq,
  simp only [int.mul_mod_left, euclidean_domain.zero_mod],},
have hp5 : q ^ 2 + (q * ↑n) ≡ 1 [ZMOD ↑n],
{ have h: q * q = q ^ 2,
  { ring,},
  rw ← h,
  rw ← mul_add,
  exact hp3,},
set x := q + ↑n with hx,
set y := q * ↑n with hy,
have hp6: q ^ 2 + y - y ≡ 1 - 0 [ZMOD ↑n],
{ apply int.modeq.sub hp5 hp4,},
simp at hp6,
exact hp6,},
```

To formalise this mathematical proof in Lean, I `split` the theorem into two parts. The first part dealt with proving the forward direction. Unfolding the definitions, I introduced q to be equal to a, and created the first hypothesis `have hp1`, which substituted b in the definition of reflexive with $q + n$.

This lead me to the creation of the first helping lemma that stated that $\gcd(a, a + b) = \gcd(a, b)$. To prove this, Professor Kevin Buzzard helped me by creating three theorems which defined the integers in terms of their absolute value and helped in proving that for any two integers m, n, $m.\gcd(n + m) = m.\gcd n$, this in turn, helped me in proving that $m.\gcd(m + n) = m.\gcd n$, by applying commutativity on addition and gcd.

Having established in my proof of `reflexive_iff_corpulent` that $q \equiv q + n$ [ZMOD $\uparrow n$] $\iff$q * (q + n) $\equiv 1$ [ZMOD $\uparrow n$], I showed that $q * (q + n) \equiv 1$ [ZMOD $\uparrow n$] is true by creating hypotheses `h1` and `h2`, which state that $q \equiv q$ [ZMOD $\uparrow n$] and that $n \equiv 0$ [ZMOD $\uparrow n$].
Showing this part was easier, as I could easily use the fact stated in previous hypothesis that `hp3` $\implies$ $q \equiv q + n$ [ZMOD $\uparrow n$], and use `h1` and `h2` to prove it..
To show that $q^2 \equiv 1$ [ZMOD $\uparrow n$] took me longer than expected. Since, I had $q * (q + n) \equiv 1$ [ZMOD $\uparrow n$], I created two new hypothesis to prove the goal. Breaking up $q * (q + n)$ gives $q * q + (q * n)$. However, since I needed q^2 instead of $q * q$, I first showed that $q * n \equiv 0$ [ZMOD $\uparrow n$], and then created a hypothesis to show that $q^2 = q * q$,
This led me to use that $q * q + (q* \uparrow n) \equiv 1$ [ZMOD $\uparrow n$], and $q^2 + (q* \uparrow n) \equiv 1$ [ZMOD $\uparrow n$].

I then subtracted from this the hypothesis `hp4`, and proved the forward direction in Lean.

Mathematical Proof of the converse:
To show : If n is corpulent, then it is reflexive :
We are given that n is corpulent, that is for all integers a with $(a, n) = 1$, we have $a^2 \equiv 1 \pmod{n}$. To show reflexive, I `split` the proof into two parts.
To show that 1. $\implies$ 2. in the definition of reflexive, notice that $a \equiv b \pmod{n}$ and $(a, b) = 1$, and that $(a, n) = 1$, so $(b, n) = 1$.
So n being corpulent $\implies$ $a^2 \equiv 1 \pmod{n}$ and $b^2 \equiv 1 \pmod{n}$.
Multiplying $a \equiv b \pmod{n}$ by b, we get $a * b \equiv b * b \pmod{n}$ which is same as $a * b \equiv 1 \pmod{n}$.
Hence, we proved that in the definition of reflexive 1. $\implies$ 2..
Now, to show that 2. $\implies$ 1.:
Given: $(a, n) = 1$ and $a^2 \equiv 1 \pmod{n}$.
Since, 2. as hypothesis now, see that $a^2 - (a * b) \equiv 0 \pmod{n}$.
This implies that $a * (a - b) \equiv 0 \pmod{n}$ and hence, n divides $a * (a - b)$.
Since, it is given that $(a, n) = 1$,
Clearly, n divides $a - b$, therefore, $a \equiv b \pmod{n}$.
Hence, proved that 2. $\implies$ 1., and therefore, if n is corpulent then it is reflexive.

Formalising the proof of the converse:
Following closely along the lines of the mathematical proof, the formalisation on Lean requires a couple of other smaller lemmas. To describe them briefly, for instance, at `htn` and `hsn`, to deal with the gcd of an integer and n, given a relation in terms of congruence modulo n between them, the following lemmas proved to be very useful and were extensively used.

```
/-- Given `gcd(s,t) = 1` and `t ≡ s [ZMOD n]`, show that `gcd(s,n) = 1`. -/
lemma gcd_sn_given_t_s {s t : ℤ} {n : ℕ} (hp : t ≡ s [ZMOD n]): (s.gcd t) = 1 →
(s.gcd n) = 1 :=
begin
  intro hst,
  --We will now proceed by contradiction.
  apply of_not_not,
  intro hk,
  set d :=  (s.gcd n) with hd, -- Letting `d` to be the `gcd (s, n)`.
  -- Since `d` is the `gcd (s,n) → d ∣ s  d ∣ n`.
```

```
  have h1: (d : ℤ) ∣ s,
  { apply int.gcd_dvd_left s n,},
  have h2: (d : ℤ) ∣ n,
  { apply int.gcd_dvd_right s n,},
  -- Since `d ∣ s → s = d * p`, for some `(p : ℤ)` and similarly, `↑n = ↑d * q`,
  --for some `(q : ℤ)`.
  have h12:  (p : ℤ), s = d * p,
  { assumption,},
  have h22 :   (q : ℤ), ↑n = ↑d * q,
  { assumption, },
  -- We have `t ≡ s [ZMOD n]`, so `↑n ∣ (t - s)` and since, `d ∣ n`, we have
  --`d ∣ (t - s)`.
  have h4 : ↑ n ∣ (t - s),
  { apply int.modeq.dvd,
    exact hp.symm,},
  have h5' : (d : ℤ) ∣ (t - s),
  { cases h22 with c hc,
    exact dvd_trans h2 h4,},
  -- The previous hypothesis imply that `d ∣ s` and `d ∣ t` but we already have
  --`h1` for former.
  have h5: (d : ℤ) ∣ t,
  { exact (dvd_iff_dvd_of_dvd_sub h5').mpr h1,},
  have h6: ↑ d ∣ ↑ (s.gcd t), -- Since `d` divides both `s` and `t`, it should
  --divide their gcd.
  { apply int.dvd_gcd h1 h5,},
  rw hst at h6, -- The `gcd (s,t) = 1`
  norm_cast at *,
  -- `d ∣ 1` so `d = 1` but we assumed it is ` 1`, hence, a contradiction!
  finish,
end

/-- Given `gcd(s,t) = 1` and `s * t ≡ 1 [ZMOD n]`, show that `gcd(s,n) = 1`. -/
@[simp]lemma gcd_sn_given_st {s t : ℤ}{n : ℕ} (hp : s * t ≡ 1 [ZMOD n]) : s.gcd n = 1 :=
begin
  -- As there was no use of `gcd(s,t) = 1` in the proof, I omitted it from the
  -- definition of lemma.
  -- We will proceed by contradiction.
  apply of_not_not, -- Assume that the `gcd(s,n)  1`.
  intro hk,
  set d :=  (s.gcd n) with hd, -- Let `d` be the `gcd (s,n)`.
  -- Clearly `d ∣ s` and `d ∣ n`.
  have h1: (d : ℤ) ∣ s,
  { apply int.gcd_dvd_left s n,},
  have h2: (d : ℤ) ∣ n,
  { apply int.gcd_dvd_right s n,},
  -- Given `s * t ≡ 1 [ZMOD n] → ↑n ∣ (s*t -1)`.
  have h3: ↑ n ∣ (s * t - 1) ,
  { apply int.modeq.dvd,
    apply hp.symm,},
  rw dvd_iff_exists_eq_mul_left at h3,
  -- With `h3` now, we can express `s * t - 1` in terms of `n`.
```

```
  have h3' : ∃ (c : ℤ), s * t - n * c = 1,
  { cases h3 with k hk,
    use k,
    rw mul_comm ↑ n k,
    linarith,},
  -- Since `d ∣ s` and `d ∣ n`, `d` should divide their linear combination.
  have h4: ∃ (c : ℤ), ↑ d ∣ (s * t - n * c),
  { cases h1 with x hx,
    cases h2 with y hy,
    cases h3 with k hk,
    rw hx,
    rw hy,
    use k,
    rw [mul_assoc, mul_assoc],
    rw ← mul_sub (↑d) (x*t) (y * k),
    simp,},
  have hs : ∃ (c : ℤ), ↑ d ∣ s * c,
  { apply dvd_mul_of_dvd_left h1,},
  have hn : ∃ (c : ℤ), ↑ d ∣ ↑ n * c,
  { apply dvd_mul_of_dvd_left h2,},
  -- Since ` c` in ℤ,  `d ∣ s * c` , `d` will divide `s * t` where `t` is an integer.
  have hs_ : ↑ d ∣ s * t,
  { apply hs,},
  have hn_ : ∃ (c : ℤ), ↑d ∣ ↑n * c,
  { tauto,},
  -- Since `d` divides `s * t - ↑n * c` so it will divide `1` as well by `h3`.
  have h5: ↑ d ∣ (1 : ℤ),
  { cases h3' with k hk,
    rw ← hk,
    specialize hn k,
    apply dvd_sub hs_ hn, },
  norm_cast at *,
  finish, -- `d ∣ 1` → `d = 1` and this imply that our claim that `d  1` is false.
end
```

To formalise the converse of the mathematical proof of `reflexive_iff_corpulent` in Lean, I introduced variables s and t to represent the variables a and b of the aforementioned mathematical proof, respectively. I then unfolded the definitions, and `split` the goal.

```
-- We will now prove the converse, i.e. `n` is corpulent → `n` is reflexive.
    { unfold corpulent,
      unfold reflexiv,
      intros hp s t hst,
      -- After unfolding the definitions of reflexive and corpulent,
      -- we observe that given `n`, `hn`, `hp`, integers `s` and `t`, and
      -- a hypothesis `hst`, we need to show two things to prove that
      --`n` is reflexive. So, the goal now is `s ≡ t [ZMOD ↑n] ⟺ s * t ≡ 1 [ZMOD ↑n]`.
      split,
      -- We will first show that given `s ≡ t [ZMOD ↑n] →  s * t ≡ 1 [ZMOD ↑n]`.
```

```
{ intro hst_mod,
  have h : t * t = t ^ 2,
  { ring,},
  have hts : t ≡ s [ZMOD ↑n],
  { apply int.modeq.symm,
    exact hst_mod,},
  have htn : t.gcd n = 1,
  { rwa gcd_sn_given_t_s hst_mod,
    rwa int.gcd_comm,},
  have ht: t * t ≡ 1 [ZMOD ↑n],
  { rw h,
    apply hp,
    exact htn,},
  have hst : s * t ≡ t * t [ZMOD ↑n],
  { apply int.modeq.mul_right t hst_mod,},
  apply int.modeq.trans hst ht,},
-- We will now show that `s * t ≡ 1 [ZMOD ↑n] → s ≡ t [ZMOD ↑n]`.
{ intro h_st,
  have hsn : s.gcd n = 1,
  { rwa gcd_sn_given_st h_st,},
  have hss : s^2 ≡ 1 [ZMOD ↑n],
  { apply hp,
    exact hsn,},
  have hs2 : s^2 = s * s,
  { ring,},
  have hs_modn : s * s ≡ 1 [ZMOD ↑n],
  { rw ← hs2,
    exact hss,},
  have hst2 : s * s - s * t ≡ 1 - 1 [ZMOD ↑n],
  { apply int.modeq.sub hs_modn h_st,},
  simp at hst2,
  have hst_sub : s * (s - t) = s * s - s * t ,
  { rw mul_sub,},
  have hp_st_sub: s * (s-t) ≡ 0 [ZMOD ↑n],
  { rw hst_sub,
    exact hst2,},
  have hn_dvd_st: ↑n ∣ s * (s - t),
  { set x := s * (s - t) with hx,
    rwa ← int.modeq_zero_iff_dvd,},
  have hns_gcd : (n : ℤ).gcd s = 1,
  { rw hsn.symm,
    exact int.gcd_comm n s,},
  have hn_dvd_st : ↑n ∣ (s - t),
  { apply int.dvd_of_dvd_mul_right_of_gcd_one hn_dvd_st hns_gcd,},
  have hts_modn: t ≡ s [ZMOD ↑n],
  { apply int.modeq_of_dvd,
    exact hn_dvd_st,},
  apply int.modeq.symm ,
  exact hts_modn,},},
```

The rough outline that the proof would follow to reach the goal is :

```
n: N
hn: 1 ≤ n
hp: {a : Z}, a.gcd ↑n = 1 → a ^ 2 ≡ 1 [ZMOD ↑n]
st: Z
hst: s.gcd t = 1
⊢ s ≡ t [ZMOD ↑n] ⟺ s * t ≡ 1 [ZMOD ↑n]
```

As is evident from the snippet, the first part is to show that

```
⊢ s * t ≡ 1 [ZMOD ↑n]
```

We are given that $s \equiv t$ [ZMOD $\uparrow n$] under the name `hst_mod` and other hypotheses as shown above. Following the mathematical proof and initially establishing that $t * t = t^2$, I concluded that $s * t \equiv t * t$ [ZMOD $\uparrow n$] using `hst`, `htn`, `hst_mod` and `ht`. Since, I have that `hst`is $s * t \equiv t * t$ [ZMOD $\uparrow n$] and `hst_mod`, which says $s \equiv t$ [ZMOD $\uparrow n$], I got the tactic mode as :

```
n: N
hn: 1 ≤ n
hp: {a : Z}, a.gcd ↑n = 1 → a ^ 2 ≡ 1 [ZMOD ↑n]
st: Z
hst: s.gcd t = 1
hst_mod: s ≡ t [ZMOD ↑n]
h: t * t = t ^ 2
hts: t ≡ s [ZMOD ↑n]
htn: t.gcd ↑n = 1
ht: t * t ≡ 1 [ZMOD ↑n]
hst: s * t ≡ t * t [ZMOD ↑n]
⊢ s * t ≡ 1 [ZMOD ↑n]
```

From here, by transitivity, the goal is proved! Now to show that:

```
n: N
hn: 1 ≤ n
hp: {a : Z}, a.gcd ↑n = 1 → a ^ 2 ≡ 1 [ZMOD ↑n]
st: Z
hst: s.gcd t = 1
h_st: s * t ≡ 1 [ZMOD ↑n]
⊢ s ≡ t [ZMOD ↑n]
```

Given `h_st` and the lemma `gcd_sn_given_st`, it is proved that $(s, n) = 1$ as shown in the snippet.

Given n is corpulent, and establishing `hss` and `hs_modn`, I subtracted `h_st` from `hs_modn` and showed that

```
↑n | s * (s - t)
```

Since, $n.\gcd s = 1$, so I get `hn_dvd_st`, which helps to show `hts_modn`, and that leads to our goal via `exact hts_modn`:

```
have hn_dvd_st : ↑n | (s - t),
have hts_modn: t ≡ s [ZMOD ↑n],
```

0.2 Mathematical Proof and Formalisation of n is corpulent $\implies m$ is corpulent for every prime power $m|n$

This proof is very straightforward and comprehensible on paper, however, it was really hard to formalise this on Lean. After unfolding my definition of corpulent and introducing a few variables, my goal was :

```
n: N
hn: 1 ≤ n
hp:  {a : Z}, a.gcd ↑n = 1 → a ^ 2 ≡ 1 [ZMOD ↑n]
pe: N
s: nat.prime p
hs: p ^ e | n
b: Z
hyp: b.gcd ↑(p ^ e) = 1
 b ^ 2 ≡ 1 [ZMOD ↑(p ^ e)]
```

I wanted to show that since $(b, (p^e)) = 1$, `hp` can be used to show that since $a^2 \equiv 1 \pmod{n}$, therefore, for $a \equiv b \pmod{(p^e)}$, and $(p^e)|n$, one can say that $a^2 \equiv 1 \pmod{(p^e)}$.
Given that I have $a \equiv b \pmod{(p^e)}$ and $a^2 \equiv 1 \pmod{(p^e)}$, I can write a in terms of b and (p^e).
Since $a \equiv b \pmod{(p^e)} \implies p^e|(a-b) \implies$
$\exists c \in Z$, such that $a = (p^e * c) + b$, now one can substitute this value of a in $a^2 \equiv 1 \pmod{(p^e)}$ and get that $b^2 \equiv 1$ [ZMOD $\uparrow (p^e)$].
Though the proof is very direct but the last step involved a lot of hypothesis to deduce the goal. As evident from the snippet of the code attached, `h_ap` implied that there exists some c, such that $a_0^2 - 1 = c* \uparrow t$ and substituting the value of a_0 in terms of b in this equation gave `h_bp`, which involved a couple more hypothesis to deduce that `h_bp` would be reduced to `h_bp_mod`.

```
n: N
have h_ap : ↑t | (a ^ 2 - 1),
```

```
{ apply int.modeq.dvd ,
exact hp'.symm, },
rw dvd_iff_exists_eq_mul_left at h_ap,
have h_bp :  (c k : Z), (c * ↑t + b)^2 - 1 = k * ↑t,
{ cases h_ap with l hl,
  cases h_ab with s hs,
  use s,
  use l,
  rwa ← hs, },
have helper :  (c k : Z), b ^ 2 - 1 = k * ↑t - c ^ 2 * ↑t ^ 2 - 2 * c * ↑t * b,
{ cases h_bp with x hx,
  cases hx with y hy,
  rw add_pow_two (x * ↑t) (b) at hy,
  use x,
  use y,
  apply eq_sub_of_add_eq',
  apply eq_sub_of_add_eq',
  rw (mul_pow x ↑t 2).symm,
  rw ← add_assoc,
  rw add_sub,
  nth_rewrite 1 (mul_assoc),
  exact hy,},
have h_bp' :  (g : Z), b^2 - 1 = g * ↑t,
{ cases helper with c hc,
  cases hc with k hk,
  use (k - c ^ 2 * ↑t - 2 * c  * b),
  rw mul_sub_right_distrib,
  rw mul_sub_right_distrib,
  rw hk,
  ring, },
rw ← dvd_iff_exists_eq_mul_left at h_bp',
have h_bp_mod : 1 ≡ b^2 [ZMOD ↑t],
{ apply int.modeq_of_dvd,
  exact h_bp',},
```

The hard part of this proof was to define `nat.coprime_lift`, which is a lemma that shows that if d divides n then the obvious reduction map units [ZMOD $\uparrow n$] $\to$ units [ZMOD $\uparrow d$] is surjective. The proof of this lemma was equally as hard on Lean as it involved dealing with `finprod`, which I have never used before. The lemmas listed below, made the approach to prove it much smoother and comprehensible.

```
lemma dvd_mem_gcd (a b : Z) : ( p : N, p.prime → ↑p | a → ¬ ↑p | b)  ⟺  a.gcd b = 1 :=
lemma dvd_mem_dvd_sum (a b c: Z) : (c | a  ¬ c | b) →  ¬ c | (a + b) :=
lemma nat.coprime_alpha_finite (n : N) (b : Z) (hn : 0 < n) :
(function.mul_support (nat.coprime_alpha_id n b)).finite :=
lemma prime_dvd_finprod { : Type} (f :  → N) (hf : (function.mul_support f).finite)
{p : N} (hp : p.prime) : p | finprod f  ⟺   t : , p | f t :=
```

However, once I had an understanding of finprod and had established these lemmas showing

that showing that a homomorphism between groups of units is surjective just followed along the Mathematical proof that
$(b+ \uparrow d * x).\gcd n = 1$, implies that if a prime divides n, then it does not divide $(b+ \uparrow d * x)$, given the definition of x and $b.gcd \uparrow d = 1$.

What I found hard and/or surprising

So far, I have always formalised a topic in Algebra and stuck with it because it seemed way more theoretical and easier. However, since this was going to be the last project of the module, I wanted to try a different field. I initially thought that Number Theory would be tough to formalise in Lean; however, I absolutely loved it! This project, for me, was way more engaging, and I liked every bit of the mathematics I formalised in this.
Though I still find it hard to logically and coherently break down each statement and prove it, I believe my thought process has immensely shifted from relying just on mathematical ideas while comprehending or proving to logically understanding each statement.
The proof of n is corpulent if m is corpulent for every prime power $m|n$ was the most challenging part of the project because I did not understand how to show surjectivity initially from units (ZMOD $\uparrow n$) $\to$ units (ZMOD $\uparrow d$) where $d|n$ and both are natural numbers. Once I had an idea of how to progress with it, the lemma `nat.coprime_lift` seemed to be the hardest to formalise because I was relying just on mathematical ideas to prove it. However, once I could logically understand the proof, it seemed way easier and more fun to prove than I imagined.
I still find it hard to prove `prime_dvd_finprod`, though it seems very much doable to me; however, I ran out of time and did could not dwell on it for long. While I still need to understand products and sums over types and subsets of types in more depth, I believe I will be able to do this, since, I have really started enjoying writing code on Lean. This module has helped me grow mathematically and think logically, always digging deeper into the most fundamental theorem, propelling me to think and link in ways I never thought I could!

What I learnt

This time, I have more things to write about my learning experience than I have about things I found hard. For starters, I can think logically! It used to be very hard for me to formalise even the most uncomplicated proofs on paper since I always thought based on mathematical ideas. However, given a proof now, I understand the direction I should proceed in. I learnt how to translate a proof on paper to Lean, even though it used to seem nearly impossible to me a month ago.
Getting used to the environment has made it easier for me to look up the necessary tactics. I finally understand that Lean is not about throwing random tactics, hoping it would prove something out of it, but is rather logically structured so that one can coherently reach the goal using appropriate tactics. Starting with the project early is the key to submitting a good project as it gives a lot of time and space for experimentation, and hence, learning is immense.
I learnt that building down from the main theorem/lemma and creating valid hypotheses makes it easier to understand the direction of the proof and avoids leading astray. In ad-

dition to all this, I learnt about various methods of doing a proof in Lean and concepts of `finprod`, `subtype`, understood coercion, and how to implement them when formalising my proof. I learnt that the goal should not be to just find a way to prove a theorem but instead to find a more efficient way to prove it. This project has been a great learning experience as I could logically break down every argument and back it up with proofs and examples. I learnt how to structure proofs when formalising in Lean, and ever since I started formalising using the `have` tactic, Lean has become a logical game! I used to dwell a lot into questioning whether the direction I am proceeding is correct or if the structure is coherent; however, I have realised that it is better to try and prove it as, ultimately, it is a lot of experimentation and the more one practices, the more they learn!