





Hey, boys,
I'm Sophia.

Hey, boys,
I'm Sophia.
You are from our
class, right?



**Hey, boys,
I'm Sophia.
You are from our
class, right?**

**You are talking “generics”.
Can we join you?**



Oh, yea! The more the merrier



An anime-style illustration of a young man with dark brown hair and a slight beard. He is wearing a green zip-up hoodie over a white t-shirt. He is standing on a wooden deck of a boat, looking towards the left. The background shows a calm sea under a clear sky.

Oh, yea! The more the merrier

*I'm Tommy,
nice to meet you!*





I'm Ricky

A boy with spiky brown hair and freckles is eating a hot dog. He is wearing a red hoodie and has a backpack. He is standing in front of a large window looking out onto a port with cargo ships and cranes. A speech bubble from his perspective says, "We were discussing about that Pepsi truck".

I'm Ricky

We were discussing
about that Pepsi truck



I'm Ricky

We were discussing
about that Pepsi truck

And what options there are to encode
it properly with generics





In the lecture Pupon
discussed inheritance





In the lecture Pupon
discussed inheritance

... using trucks as an
example, right?





**There they are,
like in the lecture!**



**There they are,
like in the lecture!**

**A cabover truck parked
near a bonneted truck!**







Here they are!



Here they are!

**I'm Luna,
by the way**

```
abstract class Truck<T>{
```

```
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){

    }
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=...;
        location(wayPoint);
    }
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=... ;
        location(wayPoint);
    }
    private Cargo<List<T>> cargo;
    private Point location;
    Truck(Point p, Cargo<List<T>> c) {
        location=p; cargo=c;
    }
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=... ;
        location(wayPoint);
    }
    private Cargo<List<T>> cargo;
    private Point location;
    Truck(Point p, Cargo<List<T>> c) {
        location=p; cargo=c;
    }
}
```

```
class Bonneted<T> extends Truck<T>{

}
```

```
class CabOver<T> extends Truck<T>{

}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=... ;
        location(wayPoint);
    }
    private Cargo<List<T>> cargo;
    private Point location;
    Truck(Point p, Cargo<List<T>> c) {
        location=p; cargo=c;
    }
}
```

```
class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, Cargo<List<T>> c) {
        super(p,c);
    }
    public int aerodynamics(){ ... }
}
```

```
class CabOver<T> extends Truck<T>{
    CabOver(Point p, Cargo<List<T>> c) {
        super(p,c);
    }
    public int aerodynamics(){ ... }
}
```

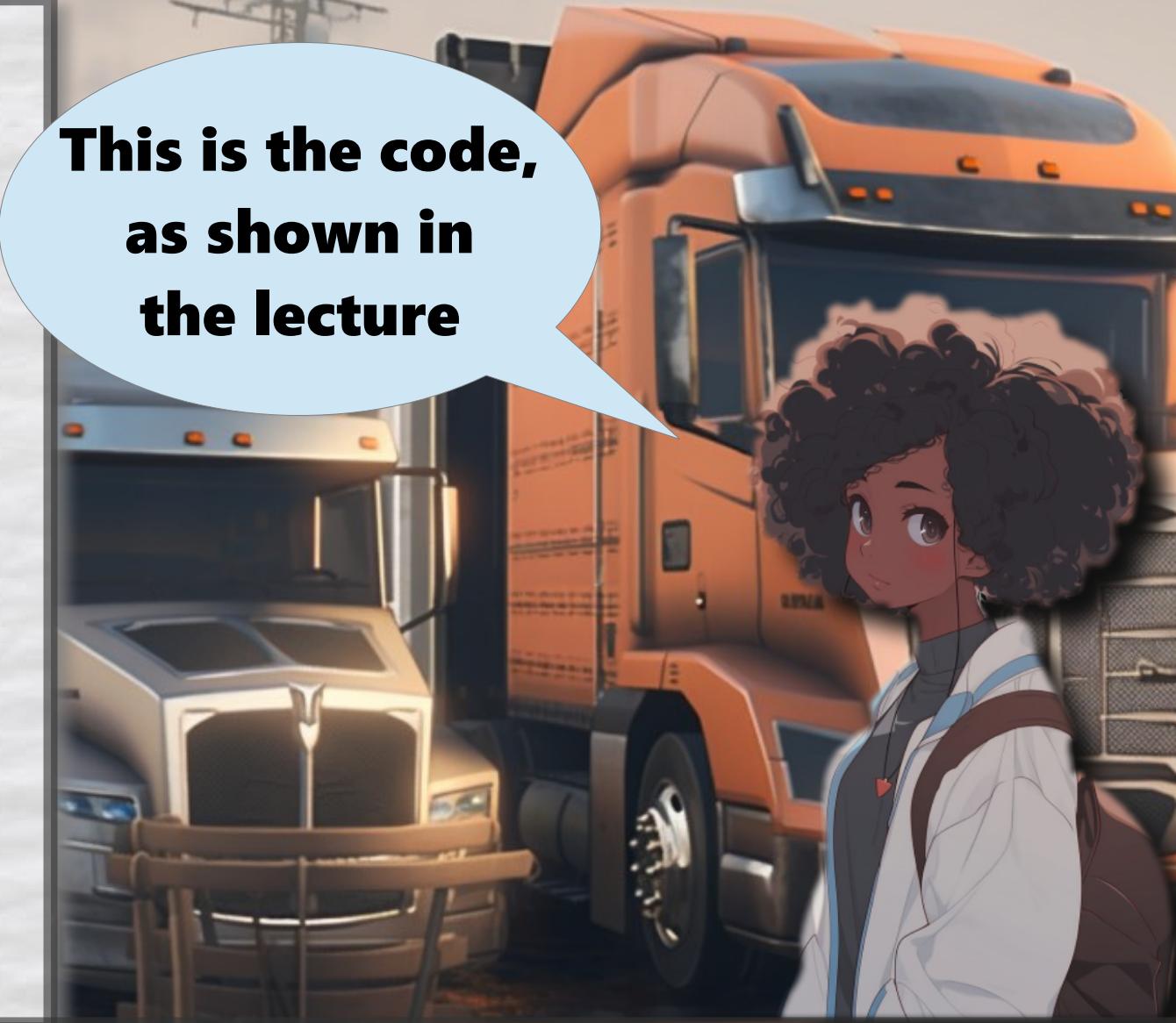


```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=... ;
        location(wayPoint);
    }
    private Cargo<List<T>> cargo;
    private Point location;
    Truck(Point p, Cargo<List<T>> c) {
        location=p; cargo=c;
    }
}
```

```
class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, Cargo<List<T>> c) {
        super(p,c);
    }
    public int aerodynamics(){ ... }
}
```

```
class CabOver<T> extends Truck<T>{
    CabOver(Point p, Cargo<List<T>> c) {
        super(p,c);
    }
    public int aerodynamics(){ ... }
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

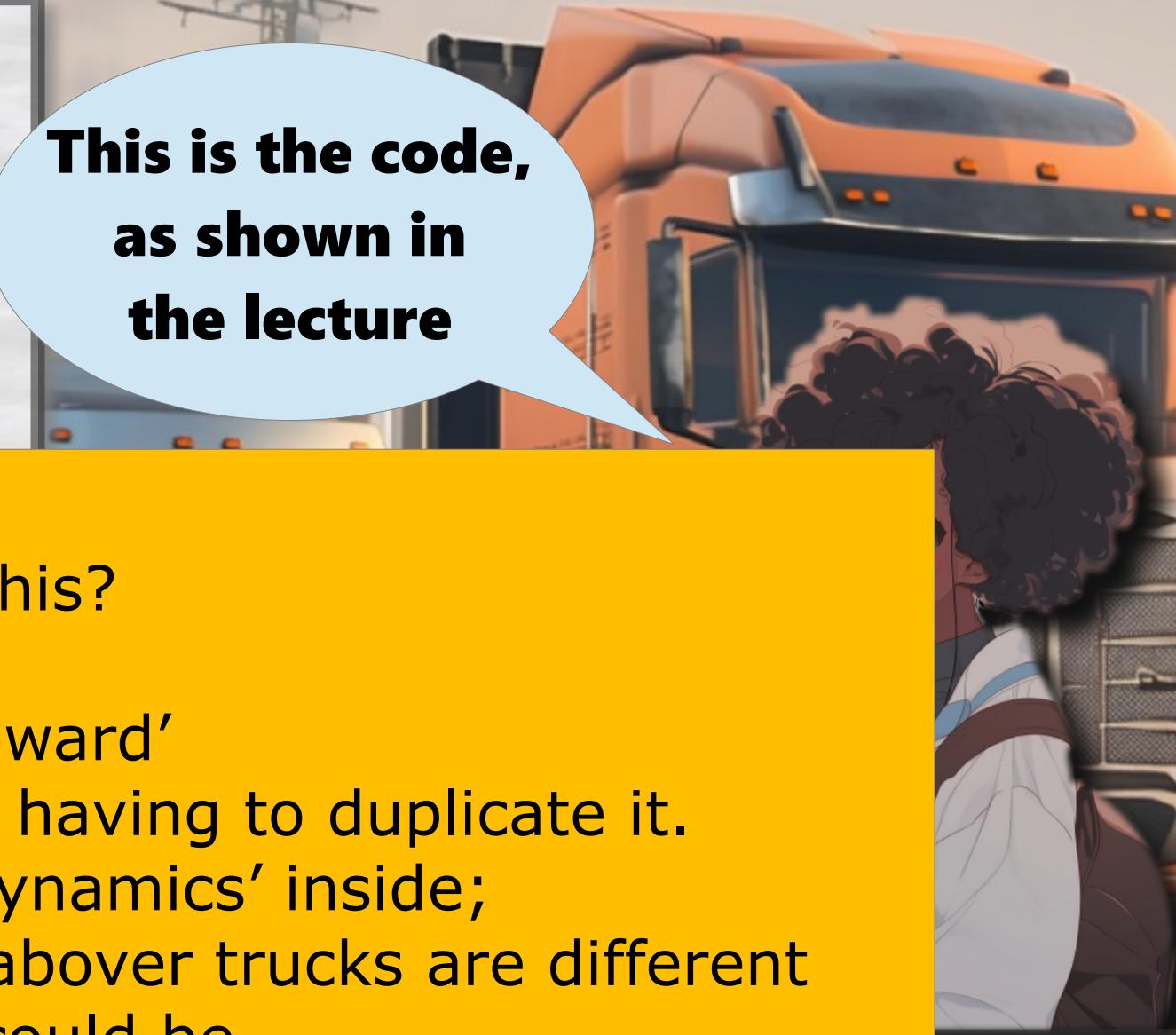
    void driveToward(Point destination, int duration){
        ...
    }
}

private class BonnetedTruck<T> extends Truck<T> {
    private Point location;
    public void location(Point p){ location=p; }

    public void driveToward(Point destination, int duration) {
        super(driveToward(destination, duration));
        aerodynamics();
    }
}

private class CabOverTruck<T> extends Truck<T> {
    private Point location;
    public void location(Point p){ location=p; }

    public void driveToward(Point destination, int duration) {
        super(driveToward(destination, duration));
        aerodynamics();
    }
}
```



Can you understand this code?

Do you think it is the best way to handle this?

The main objective is to reuse the 'driveToward' method in Bonneted and CabOver without having to duplicate it. But, 'driveToward' may have to use 'aerodynamics' inside; and the aerodynamics of bonnetted and cabover trucks are different. 'Aerodynamics' is just an example, there could be more complex differences

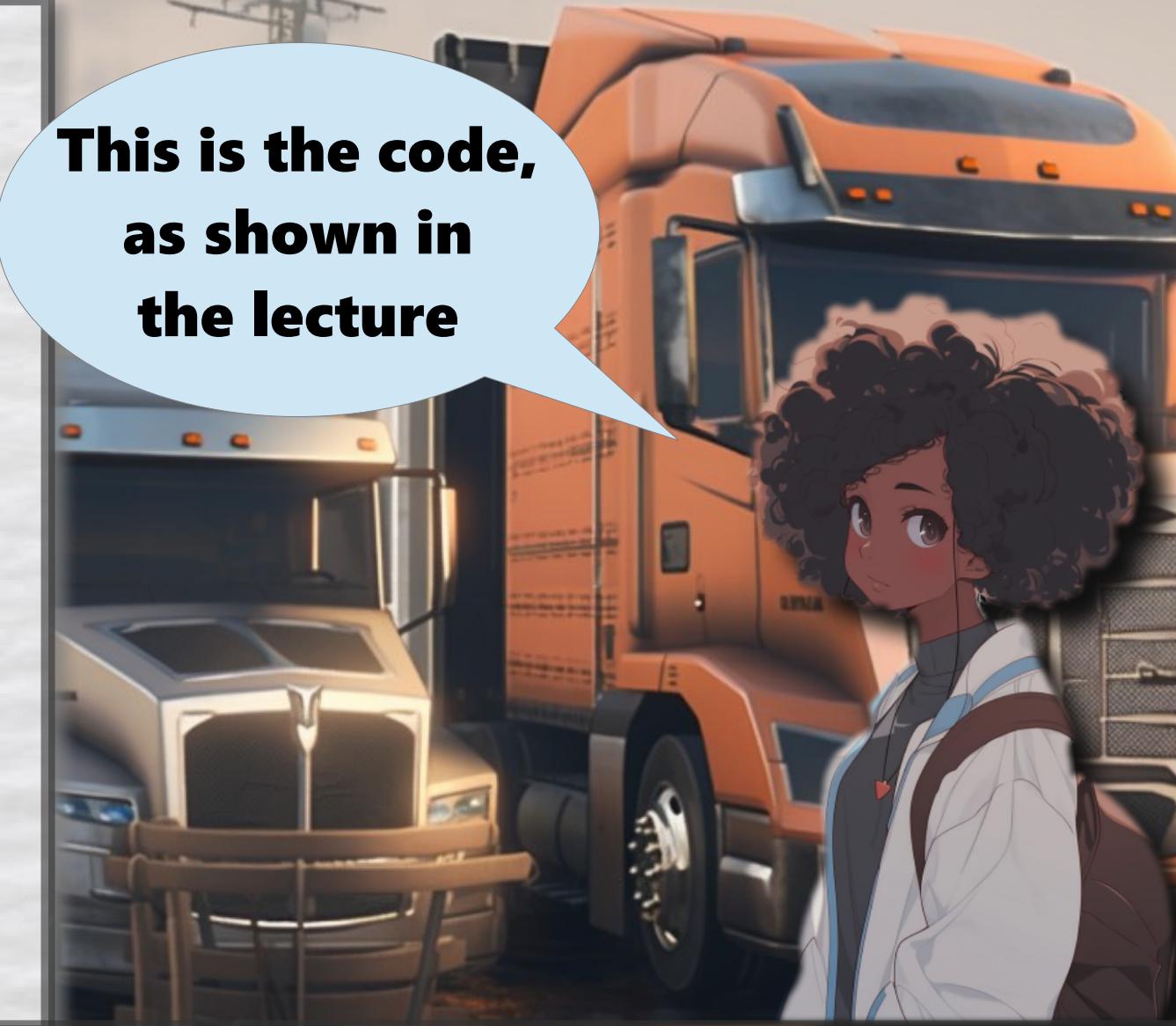
```
super(p,c);
}
public int aerodynamics(){ ... }
```

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=... ;
        location(wayPoint);
    }
    private Cargo<List<T>> cargo;
    private Point location;
    Truck(Point p, Cargo<List<T>> c) {
        location=p; cargo=c;
    }
}
```

```
class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, Cargo<List<T>> c) {
        super(p,c);
    }
    public int aerodynamics(){ ... }
}
```

```
class CabOver<T> extends Truck<T>{
    CabOver(Point p, Cargo<List<T>> c) {
        super(p,c);
    }
    public int aerodynamics(){ ... }
}
```







Wow, that is exactly
Pupon code



Wow, that is exactly
Pupon code

You must be very fast
at taking notes!





**Er... no,
I can't write
that fast.**



**Er... no,
I can't write
that fast.**

**You see, Luna has
photographic memory.**



**Er... no,
I can't write
that fast.**

**You see, Luna has
photographic memory.
She wrote it down**



**Er... no,
I can't write
that fast.**

**You see, Luna has
photographic memory.
She wrote it down
right now
just for our sake.**





You are Dany right?





**You are Dany right?
The guy that squashed
Pupon the first day.**



**You are Dany right?
The guy that squashed
Pupon the first day.**

**Surely you must have
some tricks up
your sleeves too.**



**You are Dany right?
The guy that squashed
Pupon the first day.**

**Surely you must have
some tricks up
your sleeves too.**

I'm Emma.



You are Dany right?
The guy that squashed
Pupon the first day.

Surely you must have
some tricks up
your sleeves too.

I'm Emma.
If you are planning to
get in trouble again,
do NOT involve me





Talking about trouble...

A young man with dark brown hair and a green jacket is shown from the side, looking towards the right. He is holding a single orange leaf in his right hand. The background shows a window with a view of a city skyline at sunset.

Talking about trouble...

Miss Pumpkin also
had a similar example

A young man with dark brown hair and a green zip-up hoodie is shown from the side, looking towards the right. He is holding a single orange autumn leaf between his fingers. The background is a bright, slightly overexposed outdoor scene with a railing and some foliage.

Talking about trouble...

Miss Pumpkin also
had a similar example

But the code looks
completely different!





But.. how?



But.. how?

How can you access
Pumpkin's notes?!



But.. how?

How can you access
Pumpkin's notes?!

We are Pupon students,



But.. how?

How can you access
Pumpkin's notes?!

We are Pupon students,
so UPU rules prevents
us to follow the lectures



But.. how?

How can you access
Pumpkin's notes?!

We are Pupon students,
so UPU rules prevents
us to follow the lectures
of Pumpkin



But.. how?

How can you access
Pumpkin's notes?!

We are Pupon students,
so UPU rules prevents
us to follow the lectures
of Pumpkin

Oh, I just snatched them!





You did what?



You did what?

This is very serious!



You did what?

This is very serious!



You did what?

This is very serious!

Again? After all the mess
when we stole Hanton notes?



Wait...
Wow...





Wait...
Wow...

This is not even
the first time?



Wait...
Wow...



This is not even
the first time?



This is not even
the first time?







Hey Tommy,
how did you snatch
them this time?



Hey Tommy,
how did you snatch
them this time?

I have a man
inside the other class





Wow, you sound
like a mafia boss

A boy with brown hair tied back, wearing a red hoodie, is eating an orange slice. He is looking towards the right side of the frame. A speech bubble from the left says, "But you are just saying that your friend in the other class". A speech bubble from the right says, "Wow, you sound like a mafia boss".

Wow, you sound
like a mafia boss

But you are just saying that
your friend in the other class

A boy with brown hair tied back, wearing a red hoodie, is looking up with a surprised expression. He is holding a yellow banana in his right hand. A speech bubble to his left says, "But you are just saying that your friend in the other class shared their notes with you". A speech bubble to his right says, "Wow, you sound like a mafia boss".

Wow, you sound
like a mafia boss

But you are just saying that
your friend in the other class
shared their notes with you





Those notes are going to be precious!
The final exam is coming soon

A close-up illustration of a character with dark hair and glasses, wearing a blue jacket over a red shirt. The character has a determined expression and is looking slightly upwards and to the left. The background is dark and textured.

Those notes are going to be precious!
The final exam is coming soon

It will be written half and half,
by both Pupon and Pumpkin





**Come on,
just show us the
notes already!**

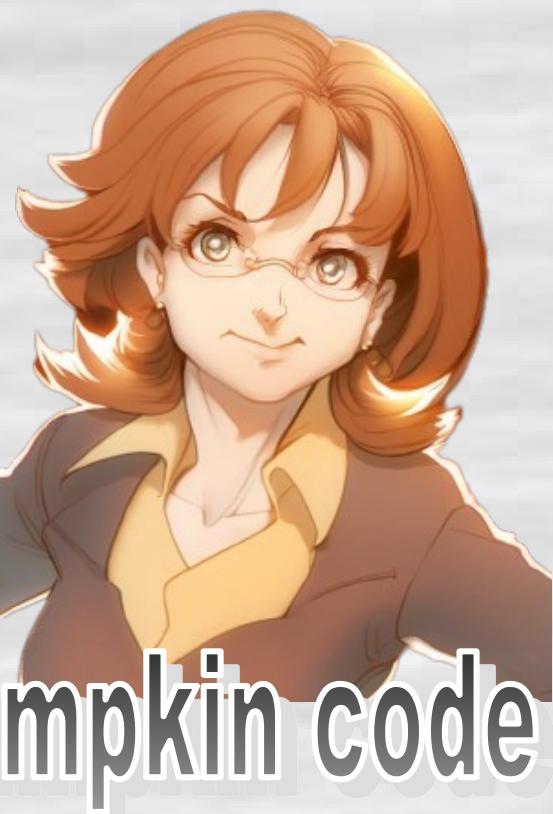




A man in a green suit is writing on a clipboard. A woman in a tan blazer is speaking to him. A speech bubble from the woman contains the text:

*Here they are!
Get ready for
the surprise!*

```
interface Truck<Self, T>{  
}  
}
```



Pumpkin code

```
interface Truck<Self, T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration) {
    }
}
```



Pumpkin code

```
interface Truck<Self, T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration) {
        ...
        Point wayPoint=...;
        return withLocation(wayPoint);
    }
}
```



Pumpkin code

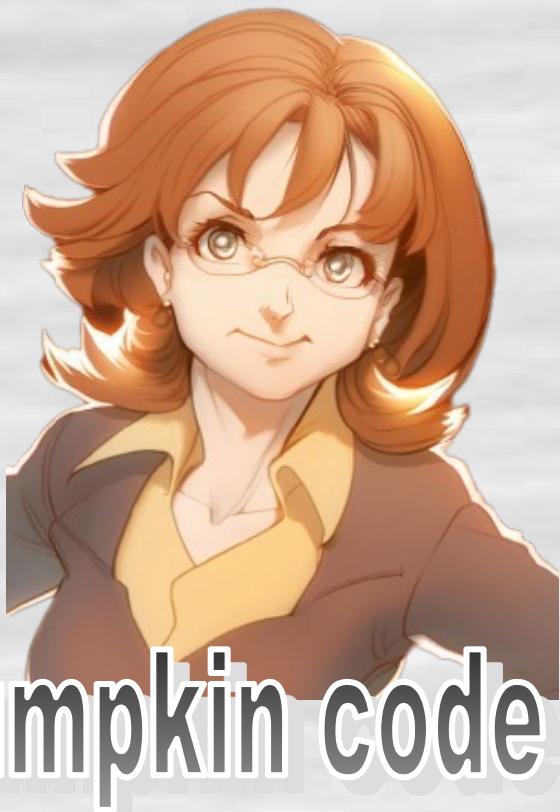
```
interface Truck<Self, T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration) {
        ...
        Point wayPoint=...;
        return withLocation(wayPoint);
    }
}
```

```
record Bonneted<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>, T>{

}
record CabOver<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>, T>{

}
```



Pumpkin code

```
interface Truck<Self, T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration) {
        ...
        Point wayPoint=... ;
        return withLocation(wayPoint);
    }
}
```



Pumpkin code

```
record Bonneted<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>, T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }

}
record CabOver<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>, T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }

}
```

```
interface Truck<Self, T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration) {
        ...
        Point wayPoint=...;
        return withLocation(wayPoint);
    }
}
```



Pumpkin code

```
record Bonneted<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>, T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
    public int aerodynamics(){ ... }
}

record CabOver<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>, T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
    public int aerodynamics(){ ... }
}
```

```
abstract class Truck<T>{  
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){

    }
}
```



Pupon code

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=...;
        location(wayPoint);
    }
}
```



Pupon code

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=...;
        location(wayPoint);
    }
    private Cargo<List<T>> cargo;
    private Point location;
    Truck(Point p, Cargo<List<T>> c){ location=p; cargo=c; }
}
```



Pupon code

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=...;
        location(wayPoint);
    }
    private Cargo<List<T>> cargo;
    private Point location;
    Truck(Point p, Cargo<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{

}

class CabOver<T> extends Truck<T>{

}
```



Pupon code

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=...;
        location(wayPoint);
    }
    private Cargo<List<T>> cargo;
    private Point location;
    Truck(Point p, Cargo<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, Cargo<List<T>> c){ super(p,c); }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, Cargo<List<T>> c){ super(p,c); }
}
```



Pupon code

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public Cargo<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=...;
        location(wayPoint);
    }
    private Cargo<List<T>> cargo;
    private Point location;
    Truck(Point p, Cargo<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, Cargo<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, Cargo<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```



Pupon code

```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}

record Bonneted<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
    public int aerodynamics(){ ... }
}

record CabOver<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
    public int aerodynamics(){ ... }
}
```



Pumpkin code

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```



Pupon code

```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}

record Bonneted<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
    public int aerodynamics(){ ... }
}

record CabOver<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
    public int aerodynamics(){ ... }
}
```



Pumpkin code

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }

    class Bonneted<T> extends Truck<T>{
        Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
        public int aerodynamics(){ ... }
    }

    class CabOver<T> extends Truck<T>{
        CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
        public int aerodynamics(){ ... }
    }
}
```



Pupon code



Wow, this is unexpected



```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}

record Bonneted<T>
(Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
public int aerodynamics(){ ... }
}

record CabOver<T>
(Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
public int aerodynamics(){ ... }
}
```



Pumpkin code

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```



Pupon code

It looks like they are using completely different languages

Wow, this is unexpected

```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}

record Bonneted<T>
(Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
public int aerodynamics(){ ... }
}

record CabOver<T>
(Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
public int aerodynamics(){ ... }
}
```

Pumpkin code

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```

Pupon code

Wow, this is unexpected

It looks like they are using completely different languages

But... how can it be?
They are modeling exactly the same concept!

```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}
```



Pumpkin code

```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
}
```



Punon code

Can you understand the difference in these two approaches?
Do you think one is better than the other?

```
record Bonne
    (Point loc)
    public Bonne()
    public int value();
}

record CabOver
    (Point loc)
    public CabOver()
    public int value();
}
```

One of these two is not as usable as it should.
Which one? Why?

Wow, this is unexpected

But... how can it be?
They are modeling exactly
the same concept!

```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}

record Bonneted<T>
(Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
    public int aerodynamics(){ ... }
}

record CabOver<T>
(Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
    public int aerodynamics(){ ... }
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```



```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}

record Bonneted<T>
(Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
    public int aerodynamics(){ ... }
}

record CabOver<T>
(Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
    public int aerodynamics(){ ... }
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```



```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}

record Bonneted<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
    public int aerodynamics(){ ... }
}

record CabOver<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
    public int aerodynamics(){ ... }
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```



We are lost for words too.



```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=...;
        return withLocation(wayPoint);
    }
}

record Bonneted<T>
(Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
    public int aerodynamics(){ ... }
}

record CabOver<T>
(Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
    public int aerodynamics(){ ... }
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

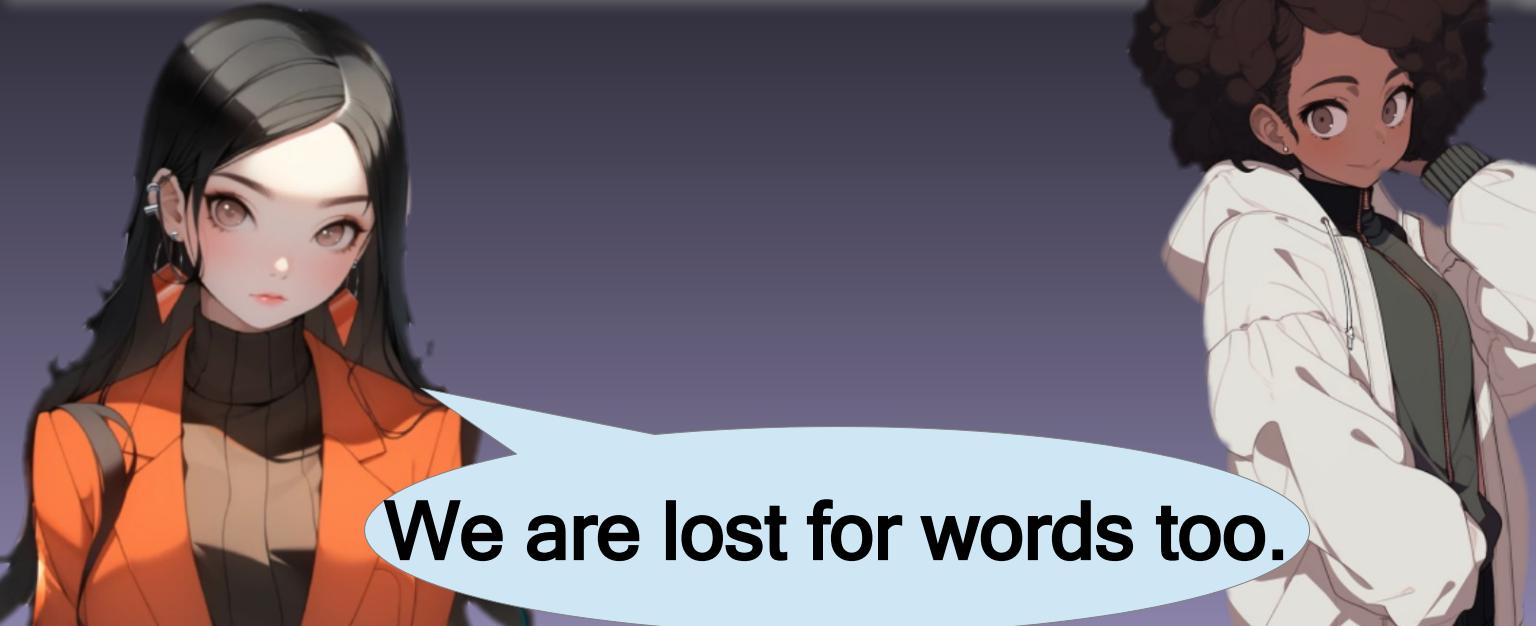
    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=...;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```



We are lost for words too.



```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}

record Bonneted<T>
(Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
    public int aerodynamics(){ ... }
}

record CabOver<T>
(Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
    public int aerodynamics(){ ... }
}
```



Pumpkin code

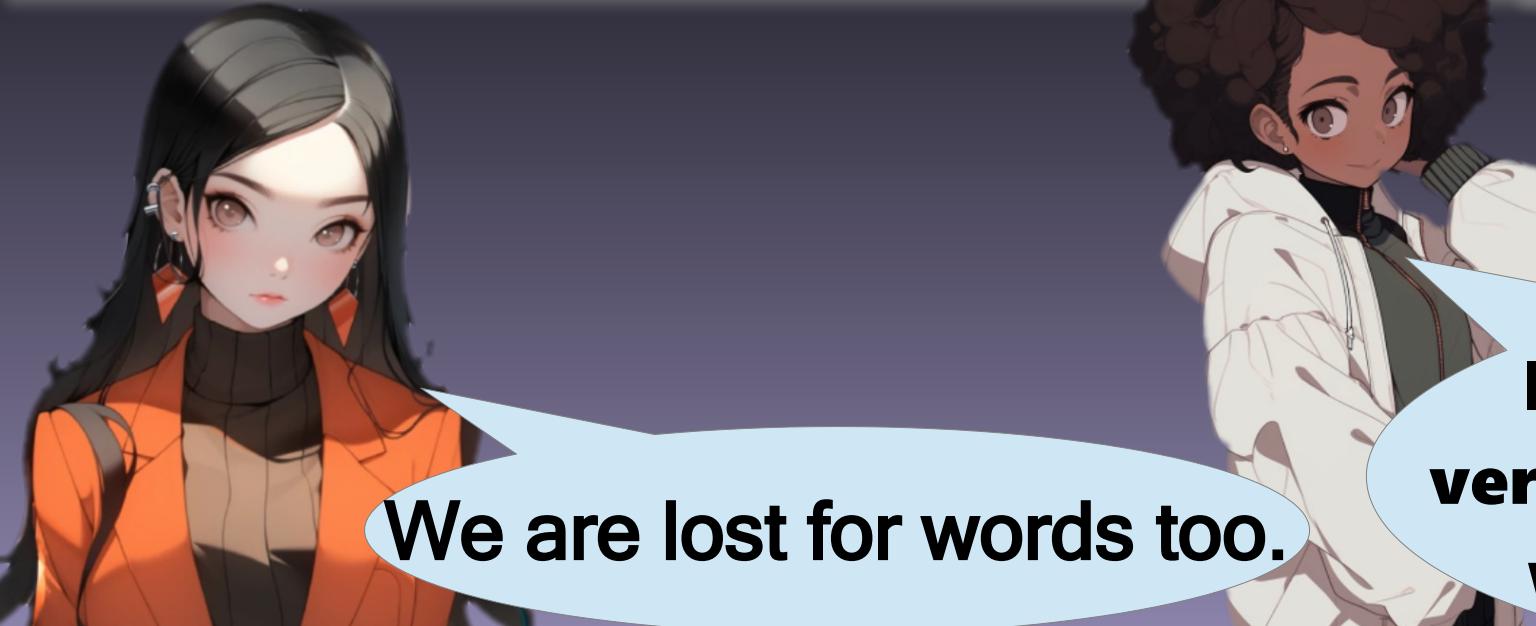
```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```

Pupon code



We are lost for words too.

I've now memorized both versions but I can't understand why they are so different!

```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}
```

```
record Bonneted<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
    public int aerodynamics(){ ... }
}

record CabOver<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
    public int aerodynamics(){ ... }
}
```



Pumpkin code

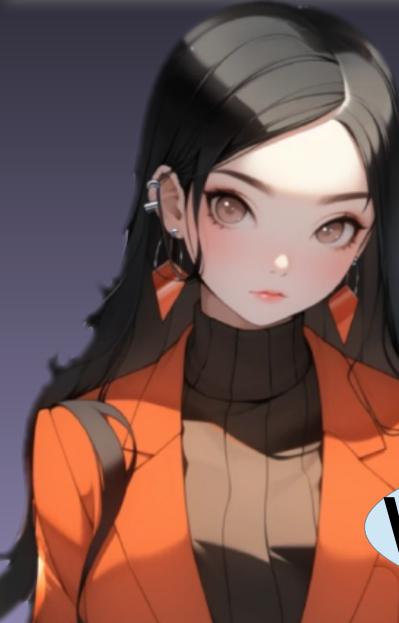
```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```

Pupon code



We are lost for words too.

I've now memorized both
versions but I can't understand
why they are so different!



```
interface Truck<Self,T>{
    int aerodynamics();
    CargoC<List<T>> cargo();
    Point location();
    Self withLocation(Point p);

    default Self driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        return withLocation(wayPoint);
    }
}
```

```
record Bonneted<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<Bonneted<T>,T>{
    public Bonneted<T> withLocation(Point p){ return new Bonneted<>(p,cargo); }
    public int aerodynamics(){ ... }
}

record CabOver<T>
    (Point location, CargoC<List<T>> cargo) implements Truck<CabOver<T>,T>{
    public CabOver<T> withLocation(Point p){ return new CabOver<>(p,cargo); }
    public int aerodynamics(){ ... }
}
```



```
abstract class Truck<T>{
    public abstract int aerodynamics();
    public CargoC<List<T>> cargo(){ return cargo; }
    public Point location(){ return location; }
    public void location(Point p){ location=p; }

    void driveToward(Point destination, int duration){
        ...
        Point wayPoint=....;
        location(wayPoint);
    }
    private CargoC<List<T>> cargo;
    private Point location;
    Truck(Point p, CargoC<List<T>> c){ location=p; cargo=c; }
}

class Bonneted<T> extends Truck<T>{
    Bonneted(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}

class CabOver<T> extends Truck<T>{
    CabOver(Point p, CargoC<List<T>> c){ super(p,c); }
    public int aerodynamics(){ ... }
}
```

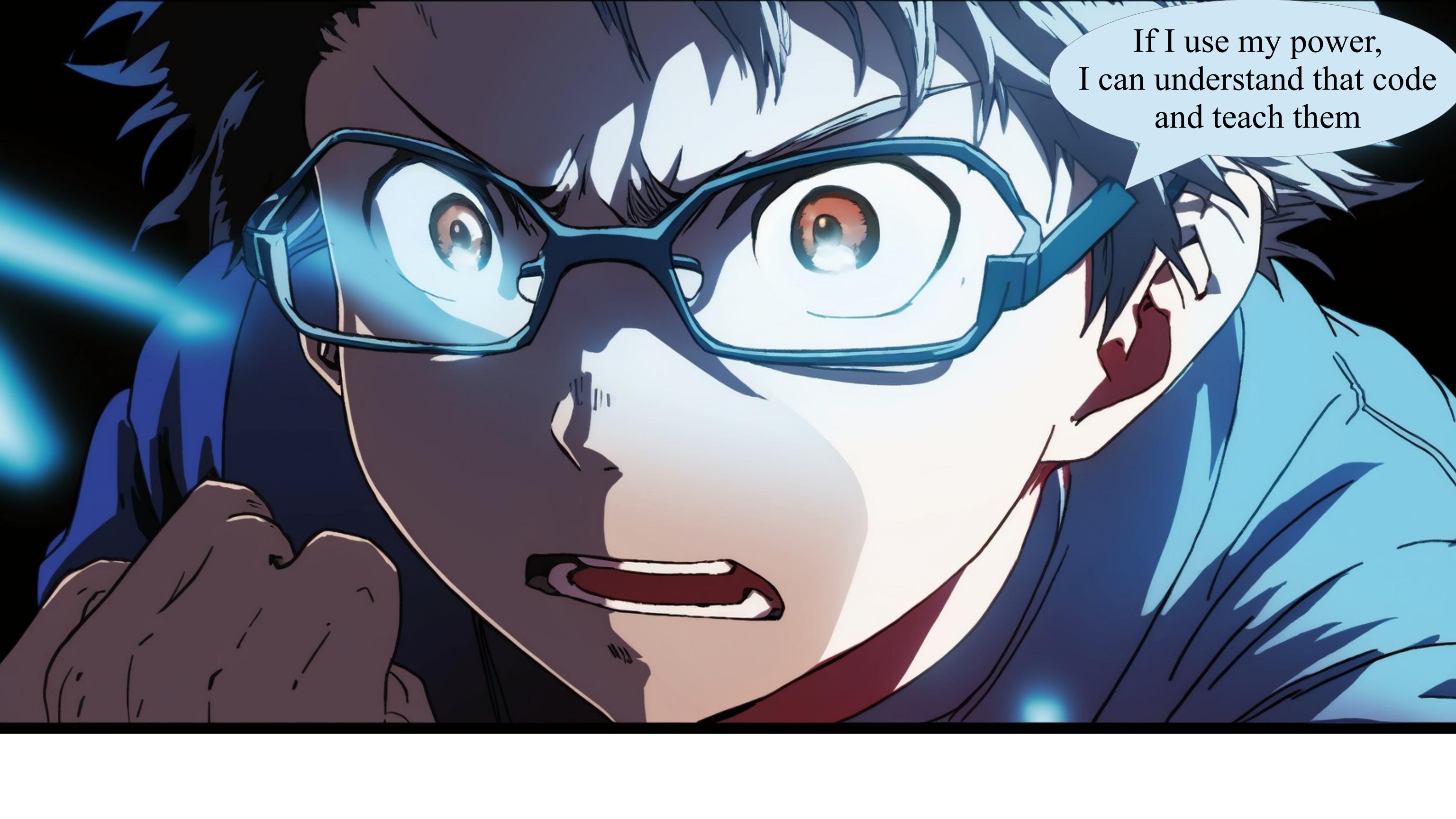


If the six of us are to pass,
we need to learn
this stuff now!



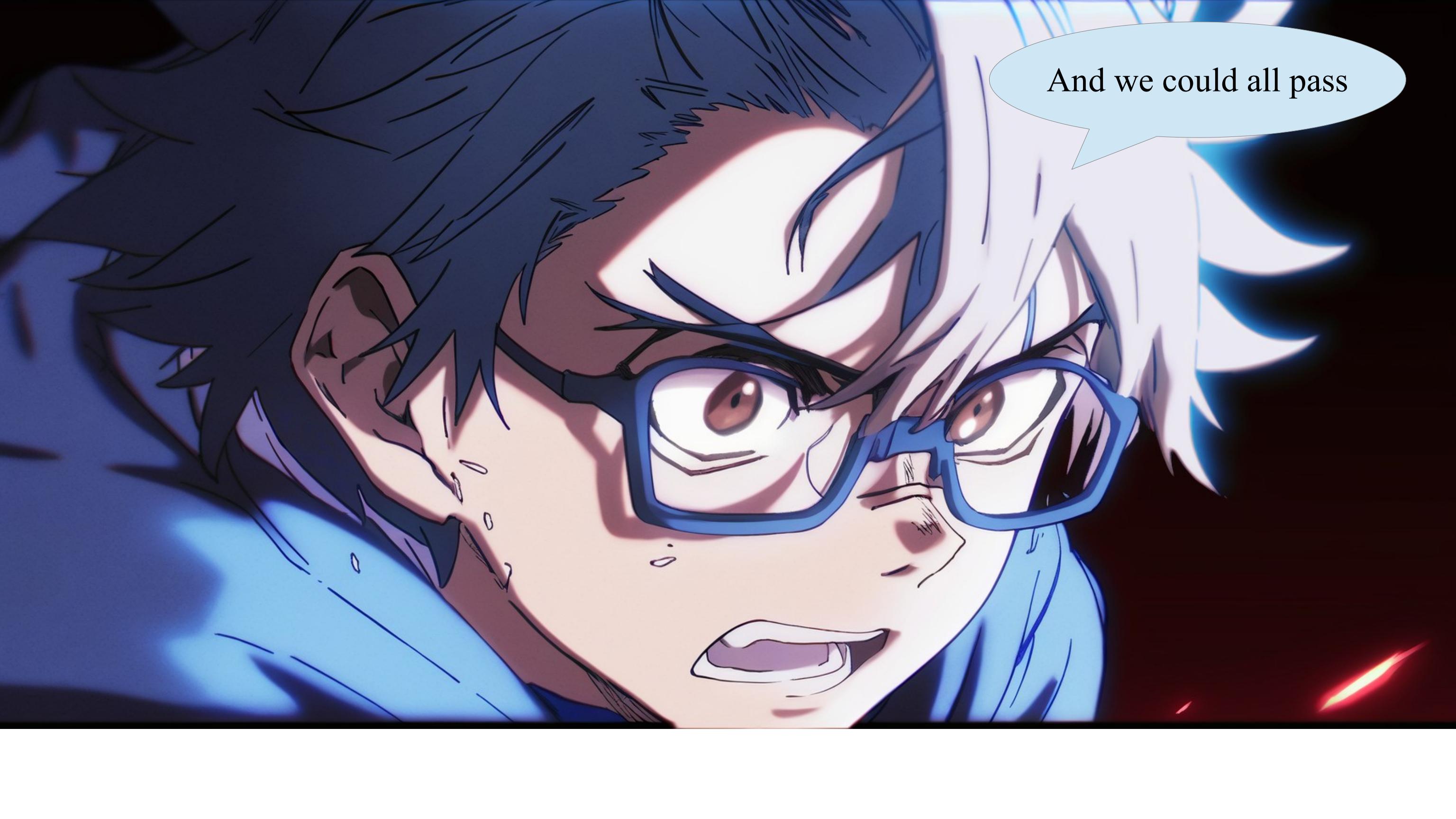
I've now memorized both
versions but I can't understand
why they are so different!

This is it.



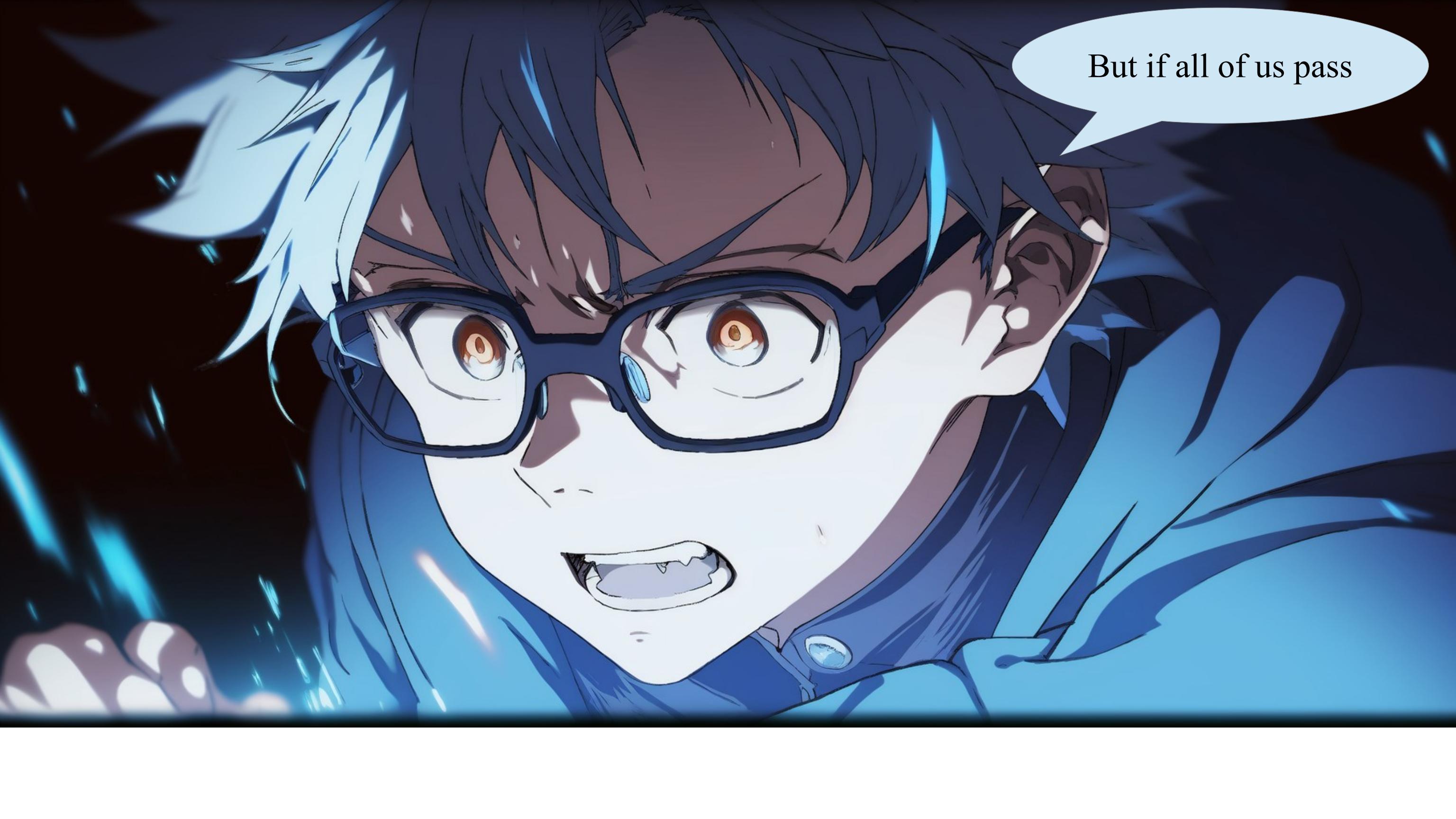
If I use my power,
I can understand that code
and teach them





And we could all pass





But if all of us pass



A close-up of an anime character with dark brown hair and red-rimmed glasses. He has a shocked expression, with wide eyes and a slightly open mouth. He is wearing a blue jacket. A speech bubble originates from his mouth.

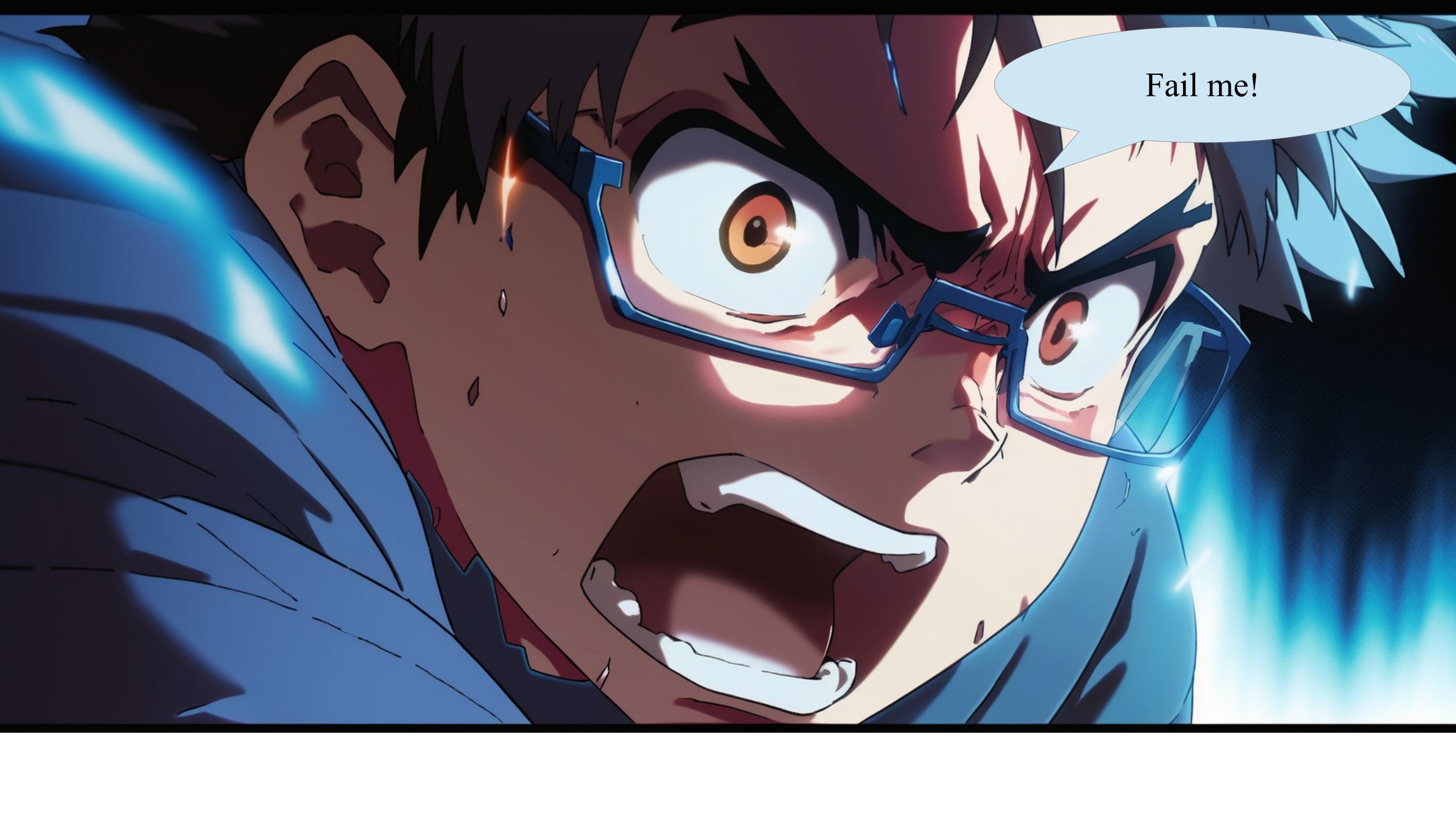
Pupon would likely
win the challenge



A close-up, dynamic shot of a man with short, spiky blue hair and brown eyes wearing black-rimmed glasses. He has a wide, open-mouthed shout or yell. His expression is intense and filled with anger or determination. The background is dark and blurred, suggesting motion or a dramatic scene.

And if he wins,
he will just ...





Fail me!





However, I cannot just ...

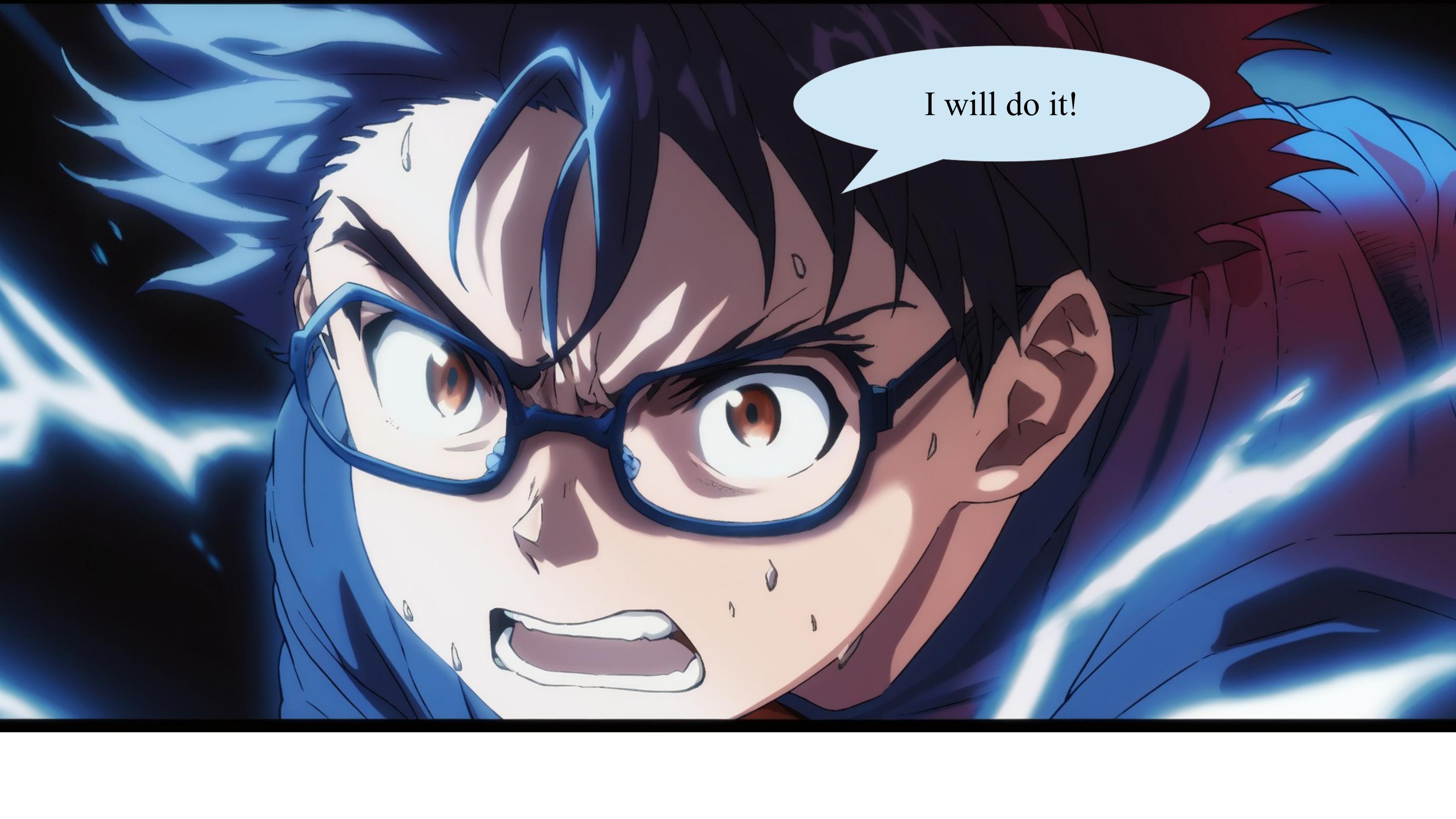


A close-up, high-angle shot of a young man with spiky blue hair and brown eyes. He wears blue-rimmed glasses and has a determined, slightly smug expression. A speech bubble originates from his mouth, containing the text.

make other people fail
so that I can pass

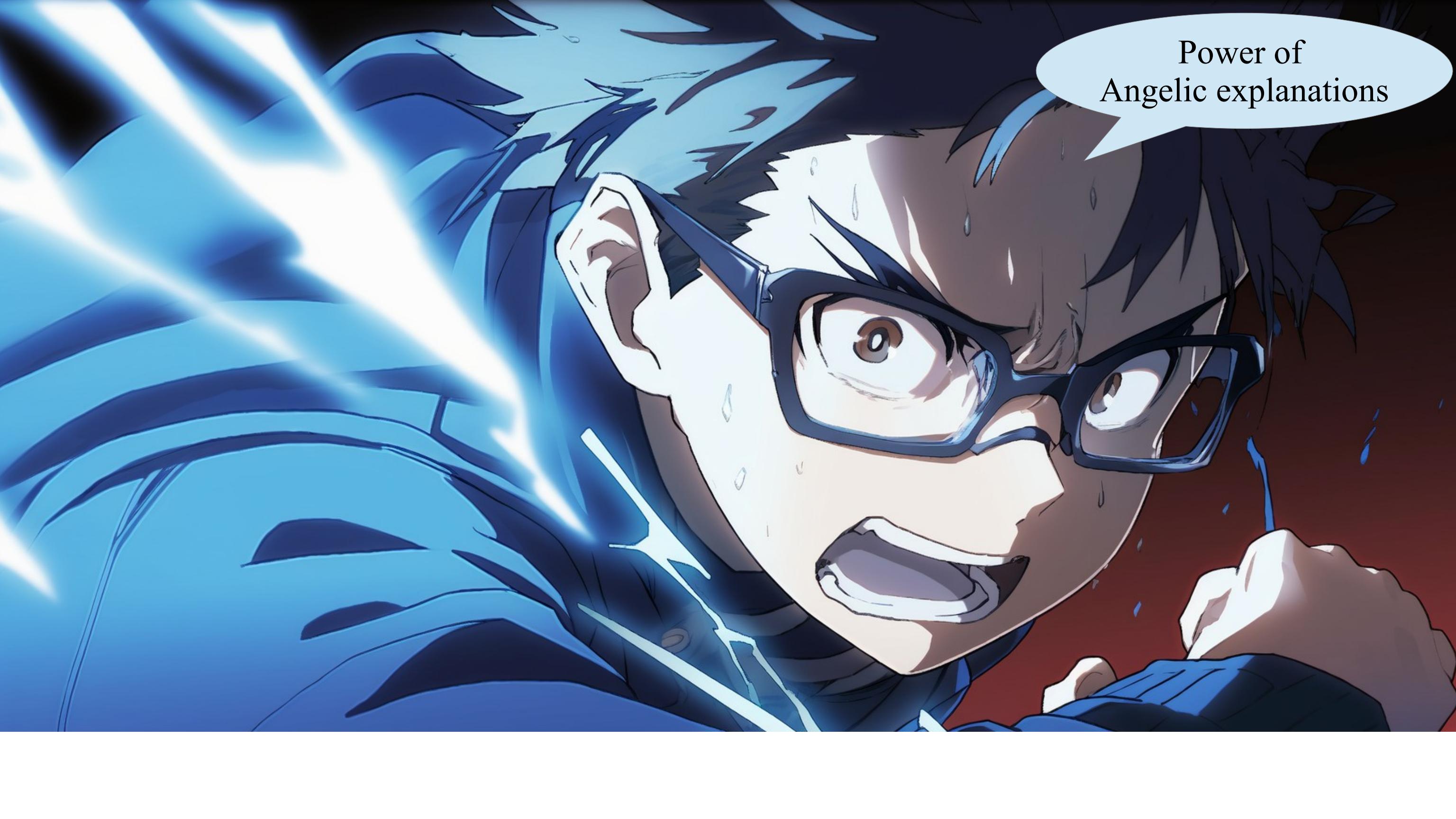






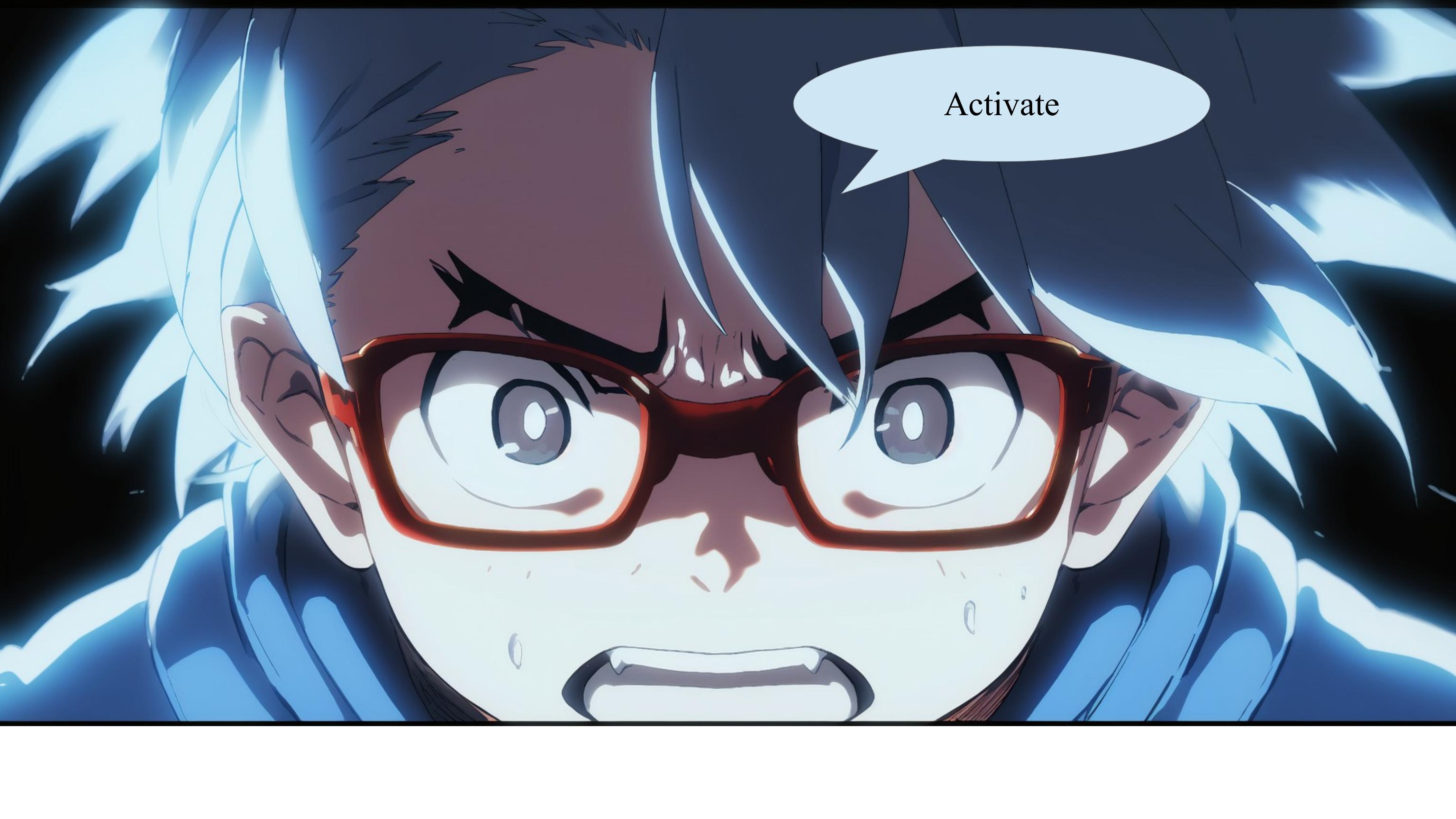
I will do it!



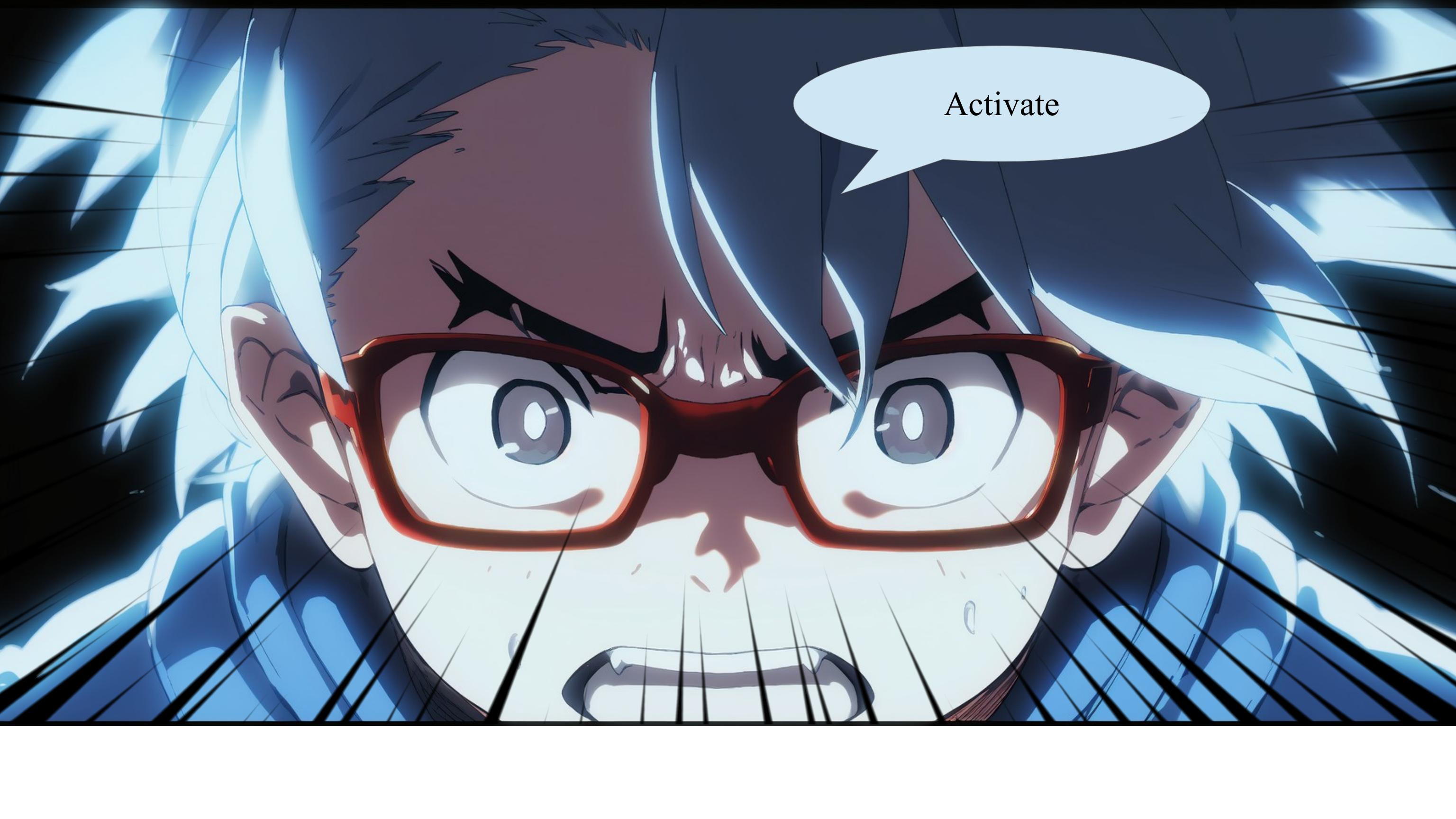


Power of
Angelic explanations



A close-up, symmetrical shot of two anime-style characters with long, spiky blue hair and red-framed glasses. They have large, wide-eyed expressions of surprise or realization. The character on the left has white highlights in their hair and is wearing a light blue jacket. The character on the right has dark blue highlights and is wearing a dark blue jacket. A speech bubble originates from the character on the right, containing the word "Activate".

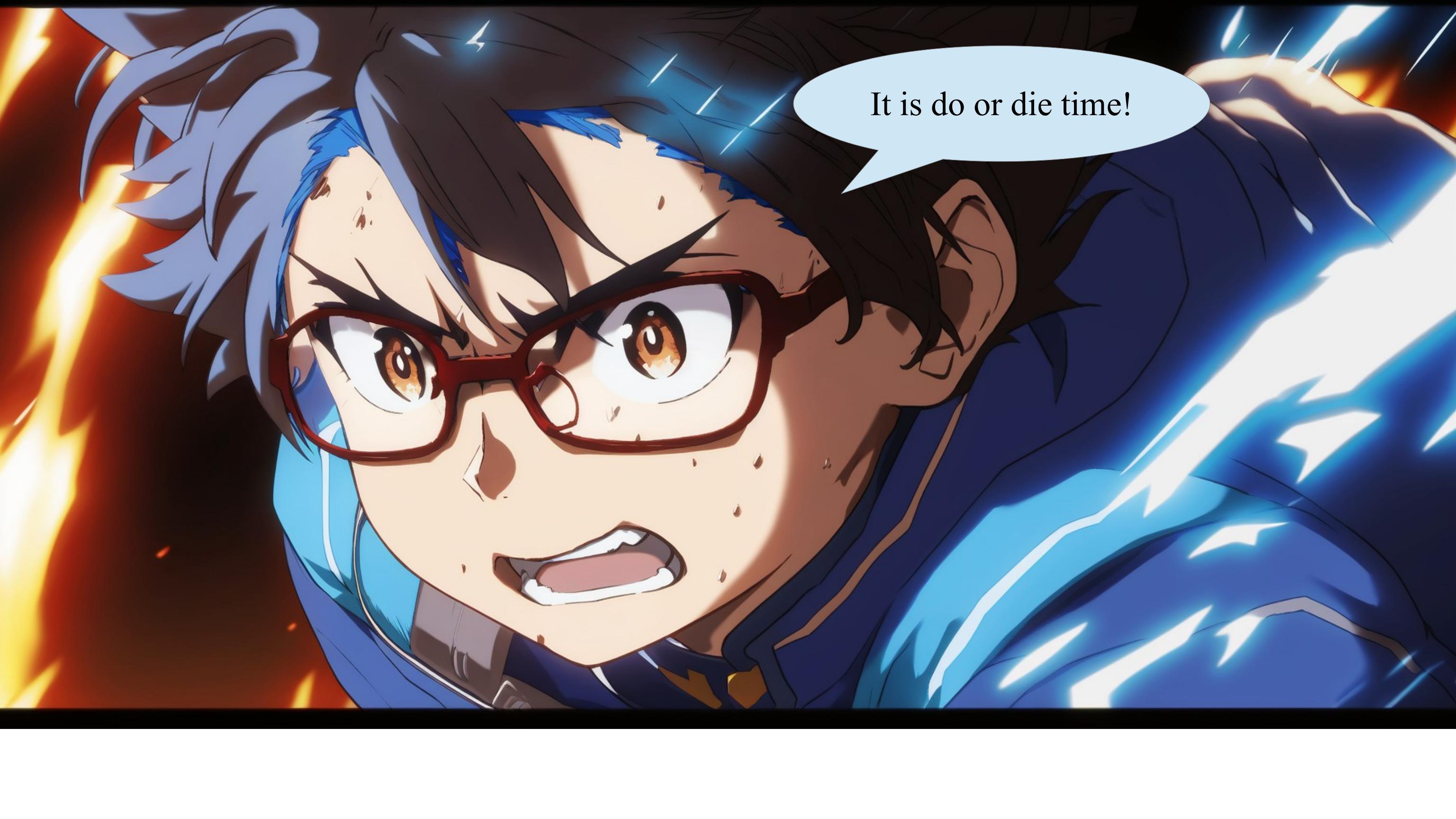
Activate



Activate







It is do or die time!