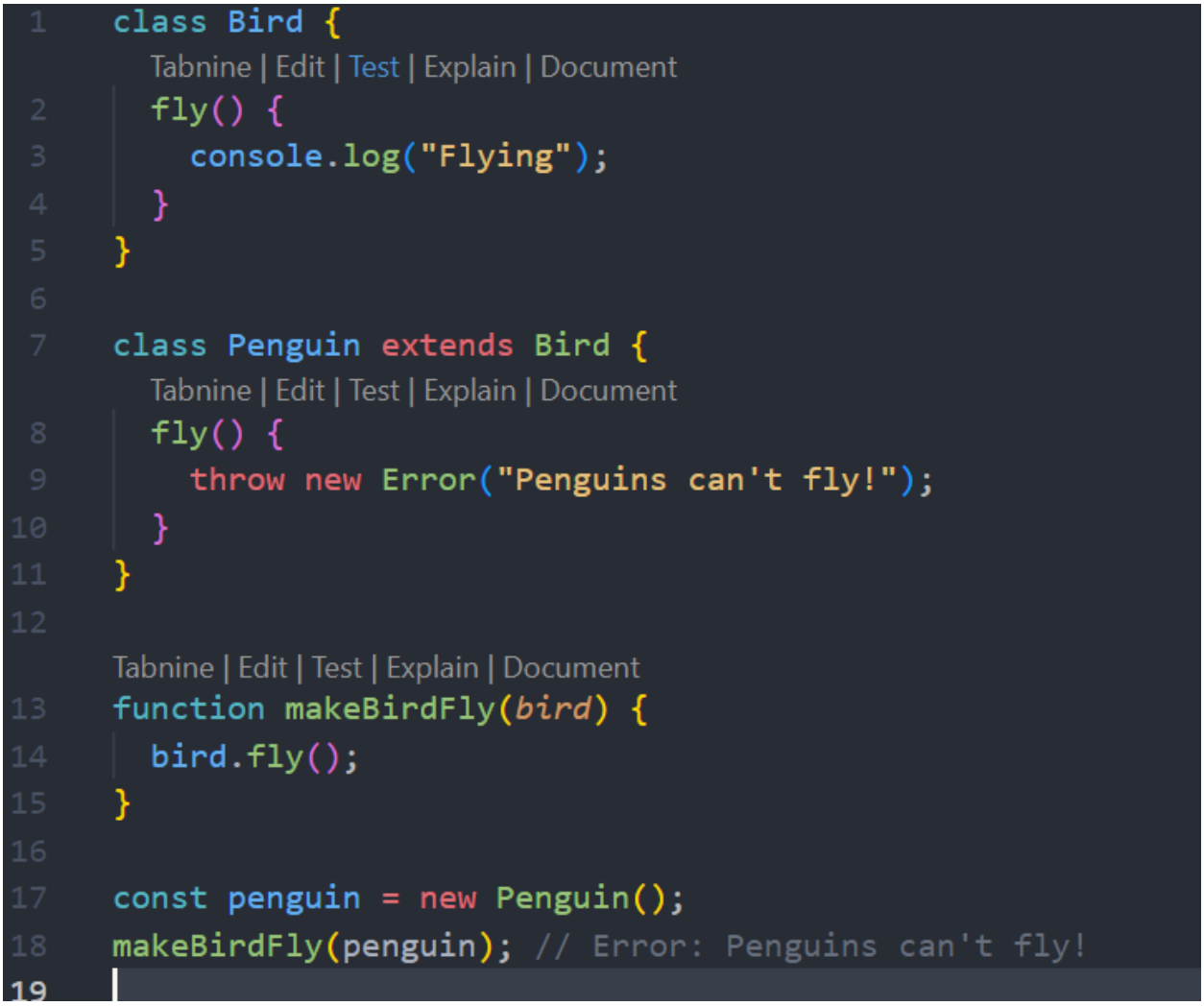
Liskov Substitution Principle (LSP)

Definition:

This principle states that objects of a superclass should be replaceable with objects of its subclass without affecting the functionality of the program.

Non-Compliant Example:



```
class Bird {
  Tabnine | Edit | Test | Explain | Document
  fly() {
    console.log("Flying");
  }
}

class Penguin extends Bird {
  Tabnine | Edit | Test | Explain | Document
  fly() {
    throw new Error("Penguins can't fly!");
  }
}

Tabnine | Edit | Test | Explain | Document
function makeBirdFly(bird) {
  bird.fly();
}

const penguin = new Penguin();
makeBirdFly(penguin); // Error: Penguins can't fly!
```

Compliant Example:

```
class Bird {
  Tabnine | Edit | Test | Explain | Document
  move() {
    console.log("I am moving");
  }
}
class FlyingBird extends Bird {
  Tabnine | Edit | Test | Explain | Document
  fly() {
    console.log("Flying");
  }
}

class Penguin extends Bird {
  Tabnine | Edit | Test | Explain | Document
  swim() {
    console.log("Swimming");
  }
}

Tabnine | Edit | Test | Explain | Document
function makeBirdMove(bird) {
  bird.move();
}

const penguin = new Penguin();
const flyingBird = new FlyingBird();

makeBirdMove(penguin); // I am moving
makeBirdMove(flyingBird); // I am moving
```

Interface Segregation Principle (ISP)

Definition:

Clients should not be forced to depend on interfaces they do not use.

Non-Compliant Example:

```
class Machine {
  Tabnine | Edit | Test | Explain | Document
  print() {}
  Tabnine | Edit | Test | Explain | Document
  scan() {}
  Tabnine | Edit | Test | Explain | Document
  fax() {}
}

class BasicPrinter extends Machine {
  Tabnine | Edit | Test | Explain | Document
  print() {
    console.log("Printing...");
  }
  Tabnine | Edit | Test | Explain | Document
  scan() {
    throw new Error("Basic Printer does not support scanning!");
  }
  Tabnine | Edit | Test | Explain | Document
  fax() {
    throw new Error("Basic Printer does not support faxing!");
  }
}

const printer = new BasicPrinter();
printer.scan(); // Error
```

Compliant Example:

```
class Printer {
  Tabnine | Edit | Test | Explain | Document
  print() {}
}

class Scanner {
  Tabnine | Edit | Test | Explain | Document
  scan() {}
}

class AdvancedPrinter extends Printer {
  Tabnine | Edit | Test | Explain | Document
  print() {
    console.log("Printing...");
  }
}

class MultiFunctionPrinter extends Printer {
  Tabnine | Edit | Test | Explain | Document
  print() {
    console.log("Printing...");
  }
  Tabnine | Edit | Test | Explain | Document
  scan() {
    console.log("Scanning...");
  }
}

const printer = new AdvancedPrinter();
printer.print(); // Printing...
```

Dependency Inversion Principle (DIP)

Definition:

High-level modules should not depend on low-level modules; both should depend on abstractions.

Non-Compliant Example:

```
class LightBulb {
  Tabnine | Edit | Test | Explain | Document
  turnOn() {
    console.log("LightBulb is on");
  }
}

class Switch {
  constructor() {
    this.lightBulb = new LightBulb();
  }

  Tabnine | Edit | Test | Explain | Document
  toggle() {
    this.lightBulb.turnOn();
  }
}

const switchObj = new Switch();
switchObj.toggle(); // LightBulb is on
```

Compliant Example:

```
class Device {
  Tabnine | Edit | Test | Explain | Document
  turnOn() {}
}
class LightBulb extends Device {
  Tabnine | Edit | Test | Explain | Document
  turnOn() {
    console.log("LightBulb is on");
  }
}
class Fan extends Device {
  Tabnine | Edit | Test | Explain | Document
  turnOn() {
    console.log("Fan is on");
  }
}
class Switch {
  constructor(device) {
    this.device = device;
  }

  Tabnine | Edit | Test | Explain | Document
  toggle() {
    this.device.turnOn();
  }
}

const lightBulb = new LightBulb();
const fan = new Fan();

const lightSwitch = new Switch(lightBulb);
lightSwitch.toggle(); // LightBulb is on

const fanSwitch = new Switch(fan);
fanSwitch.toggle(); // Fan is on
```