**Here is how I understood not just what virtual functions are, but why they're required:**

**Let's say you have these two classes:**

**class Animal**

**{**

**public:**

**void eat() { std::cout << "I'm eating generic food."; }**

**};**

**class Cat: public Animal**

**{**

**public:**

**void eat () { std::cout << "I'm eating a rat."; }**

**};**

**In your main function:**

**Animal \*animal = new Animal;**

**Cat \*cat = new Cat;**

**animal->eat (); // Outputs: "I'm eating generic food."**

**cat->eat (); // Outputs: "I'm eating a rat."**

**So far so good, right? Animals eat generic food, cats eat rats, all without virtual.**

**Let's change it a little now so that eat() is called via an intermediate function (a trivial function just for this example):**

**// This can go at the top of the main.cpp file**

**void func(Animal \*xyz) { xyz->eat(); }**

**Now our main function is:**

**Animal \*animal = new Animal;**

**Cat \*cat = new Cat;**

**func(animal); // Outputs: "I'm eating generic food."**

**func(cat); // Outputs: "I'm eating generic food."**

**Uh oh... we passed a Cat into func(), but it won't eat rats. Should you overload func() so it takes a Cat\*? If you have to derive more animals from Animal they would all need their own func().**

**The solution is to make eat() from the Animal class a virtual function:**

**class Animal**

**{**

**public:**

**virtual void eat () { std::cout << "I'm eating generic food."; }**

**};**