What is difference between `mut a: &T` and `a: &mut T`?

Function parameters and let bindings in Rust are proper patterns, like those at the left of => in match (except that let and parameter patterns must be irrefutable, that is, they must always match). mut a is just a part of pattern syntax and it means that a is a mutable binding. &mut T/&T, on the other hand, is a type - mutable or immutable reference.

There are four possible combinations of mut in references and patterns:

```
    a: &T       // immutable binding of immutable reference
mut a: &T       // mutable binding of immutable reference
    a: &mut T   // immutable binding of mutable reference
mut a: &mut T   // mutable binding of mutable reference
```

The first variant is absolutely immutable (without taking internal mutability of Cell and such into account) - you can neither change what a points to nor the object it currently references.

The second variant allows you to change a to point somewhere else but it doesn't allow you to change the object it points to.

The third variant does not allow to change a to point to something else but it allows mutating the value it references.

And the last variant allows both changing a to reference something else and mutating the value this reference is currently pointing at.

Taking the above into account you can see where mut a: &T can be used. For example, you can write a search of a part of a string in a loop for the further usage like this:

```
let mut s: &str = source;
loop {
    // ... whatever
    s = &source[i..j];
}
// use the found s here
```