

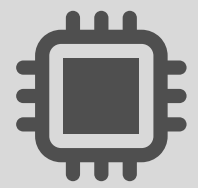


Can you explain the type error in the Haskell code:

```
mapList :: (a → b) → [a] → [b]
```

```
mapList func [] = []
```

```
mapList func (x:xs) = func x : mapList func xs (func x)
```



Sure! This is the Haskell code involving a type error:

```
mapList :: (a → b) → [a] → [b]
```

```
mapList func [] = []
```

```
mapList func (x:xs) = func x : mapList func xs (func x)
```

The mapList function is attempting to implement a basic map operation: it takes a function of $a \rightarrow b$, and applies it to each element of a list of a , producing a list of b .

However, there's a type error in the recursive call: `mapList func xs (func x)`. We are giving it three arguments where it only expects two. This results in an infinite type situation because Haskell tries to unify the type a with the type $(a \rightarrow b) \rightarrow [a] \rightarrow [b]$, which results in an infinite loop of types, and Haskell cannot resolve this.