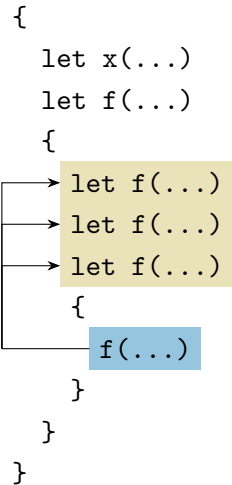
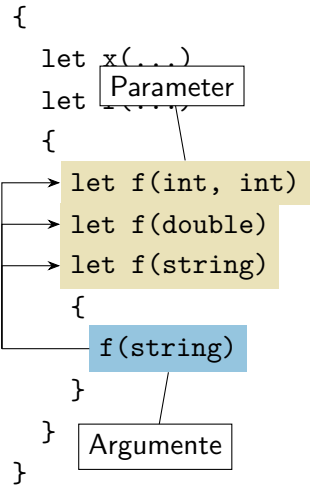


```
{  
  let x(...)  
  let f(...)  
  {  
    let f(...)  
    let f(...)  
    let f(...)  
    {  
      f(...)  
    }  
  }  
}
```

```
{  
  let x(...)  
→ let f(...)  
  {  
    let f(...)  
    let f(...)  
    let f(...)  
    {  
      f(...)  
    }  
  }  
}
```

The diagram illustrates the lexical scoping resolution for the function `f(...)`. It shows a nested structure of function declarations and calls. The first `let f(...)` is the outermost, followed by two more `let f(...)` declarations inside a block. The final `f(...)` is a call to the function. Arrows indicate the resolution path: three arrows point from the `let f(...)` declarations to a yellow box containing `let f(...)`, and one arrow points from the `f(...)` call to a blue box containing `f(...)`. A vertical dashed line is on the left side of the code.





```
{  
  let x(...)  
  let f(...)  
  {  
    let f(int, int)  
    let f(double)  
    let f(string)  
    {  
      f(string)  
    }  
  }  
}
```

The diagram illustrates function calls within a block. A vertical line on the left side of the code block has two arrows pointing to the right. The top arrow points to the definition of `f(double)`, and the bottom arrow points to the call `f(string)` inside the nested block. This indicates that the function `f` is being called with different arguments within the same scope.

```
{  
  let x(...)  
  let f(...)  
  {  
    let f(int, int)  
    let f(double)  
    let f(string)  
    {  
      f(string)  
    }  
  }  
}
```

The diagram illustrates a function call within a function definition. A line originates from the `f(string)` call inside the inner function block and points back to the `let f(string)` definition, indicating a recursive or self-call.