Here's my solution (I'm just showing `forward`method of the `Add`class):

```
def forward(self):
    x_value = self.inbound_nodes[0].value
    y_value = self.inbound_nodes[1].value
    self.value = x_value + y_value
```

While this looks simple, I want to show you why I used `x_value` and `y_value`from the `inbound_nodes` array. Let's take a look at the start with `Node`'s constructor:

```
class Node(object):
    def __init__(self, inbound_nodes=[]):
        # Node(s) from which this Node receives values.
        self.inbound_nodes = inbound_nodes
        # Removed everything else for brevity.
```

`inbound_nodes` are set when the `Node` is instantiated.

Of course, you weren't using `Node` directly, rather you used `Add`, which is a subclass of `Node`. `Add`'s constructor is responsible for passing the `inbound_nodes` to `Node`, which happens here:

```
class Add(Node):
    def __init__(self, x, y):
         Node.__init__(self, [x, y]) # calls Node's constructor
    ...
```

Lastly, there's the question of why `node.value` holds the value of the inputs. For each node of the `Input()` class, the nodes are set directly when you run `topological_sort`:

```
def topological_sort(feed_dict):
    ...
    if isinstance(n, Input):
        n.value = feed_dict[n]
```

```
        ...
```

For other classes, the value of node.value is set in the forward pass:

```
def forward_pass(output_node, sorted_nodes):
    ...
    for n in sorted_nodes:
        n.forward()
    ...
```

And that's it for addition!

Keep going to make `MiniFlow` more capable.

Bonus Challenges!

These are **ungraded**challenges as they are more of a test of your Python skills than neural network skills.

1. Can you make `Add` accept any number of inputs? Eg. `Add(x, y, z)`.
2. Can you make a `Mul` class that multiplies *n*inputs?