Tree Traversal Patterns

- General pattern for recursive depth-first traversing General Trees:

```
to traverseDF(node):
    process node
    foreach child of node
        traverseDF (child)
```

Need to modify to:
- stop early
- return values
- use the depth
- …

- General pattern for iterative traversing General Trees:

```
to traverseBF(node)
    put node on queue
    while (queue is not empty)
        dequeue node from queue
        process node
        foreach child of node
            put child on queue
```

```
to traverseDF(node)
    put node on stack
    while (stack is not empty)
        pop node from stack
        process node
        foreach child of node
            push child on stack
```

Tree Traversal Patterns

- General pattern for recursive depth-first traversing General Trees: passing depth and other information **down** the tree.

```
to traverseDF(node)
    traverse(node, 0, info)

to traverseDF(node, depth, info):
    process node at depth with info
    foreach child of node
        traverseDF (child, depth+1, addto(info))
```

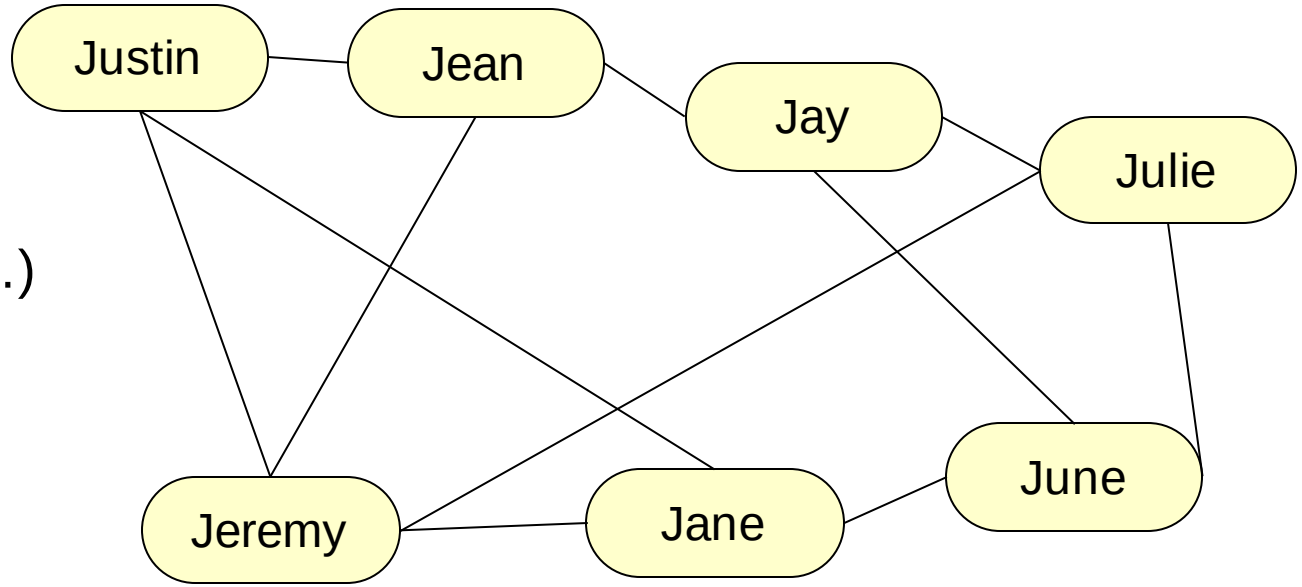
Tree Traversal Patterns

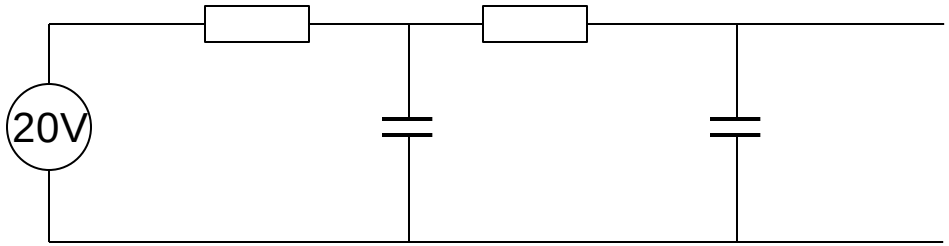
- General pattern for recursive depth-first traversing General Trees: passing information back **up** the tree.

```
to traverseDF(node):
    ans = …. process node
    foreach child of node
        childAns = traverseDF (child)
        combine childAns into ans
    return ans
```

Graph/Network Structured Data

- Examples:
  - social networks
  - circuit diagrams
  - network structures (communication, airline,…)
  - road maps,
  - database structure diagrams
  - … whenever there are relationships between data items.

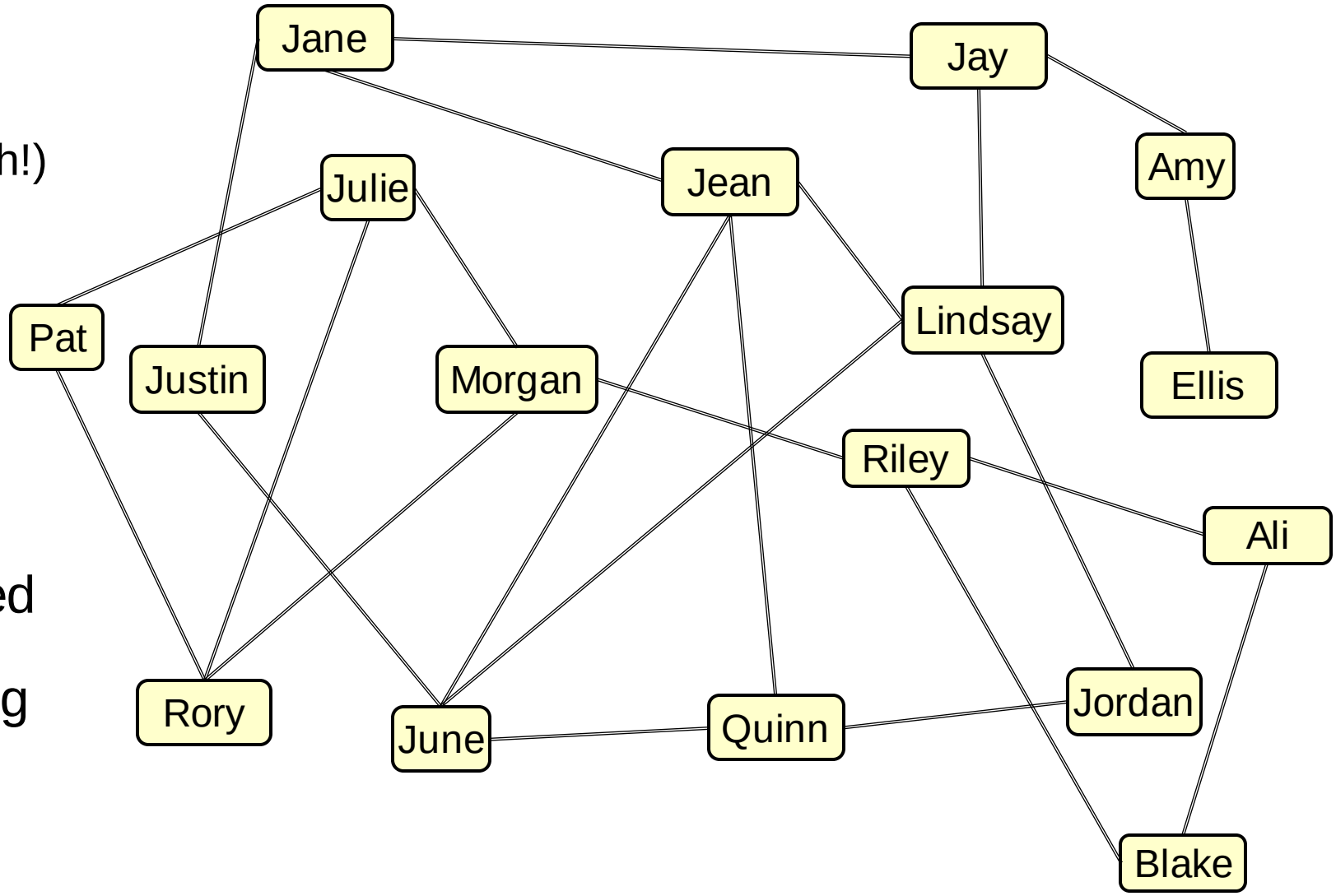




- Nodes and links, (or vertices and edges, if you are a mathematician)

Graphs

- Graphs are like trees:
  - Nodes and Links (edges)
  - (Trees are a special kind of graph!)
- Nodes have neighbours
  - rather than children
- Graphs don't have a "root"
  - (typically)
- Graphs may not be connected
- Can traverse a graph, starting at node, but
- **Graphs have cycles!**
- Lots of varieties of graphs – this one is the simplest.

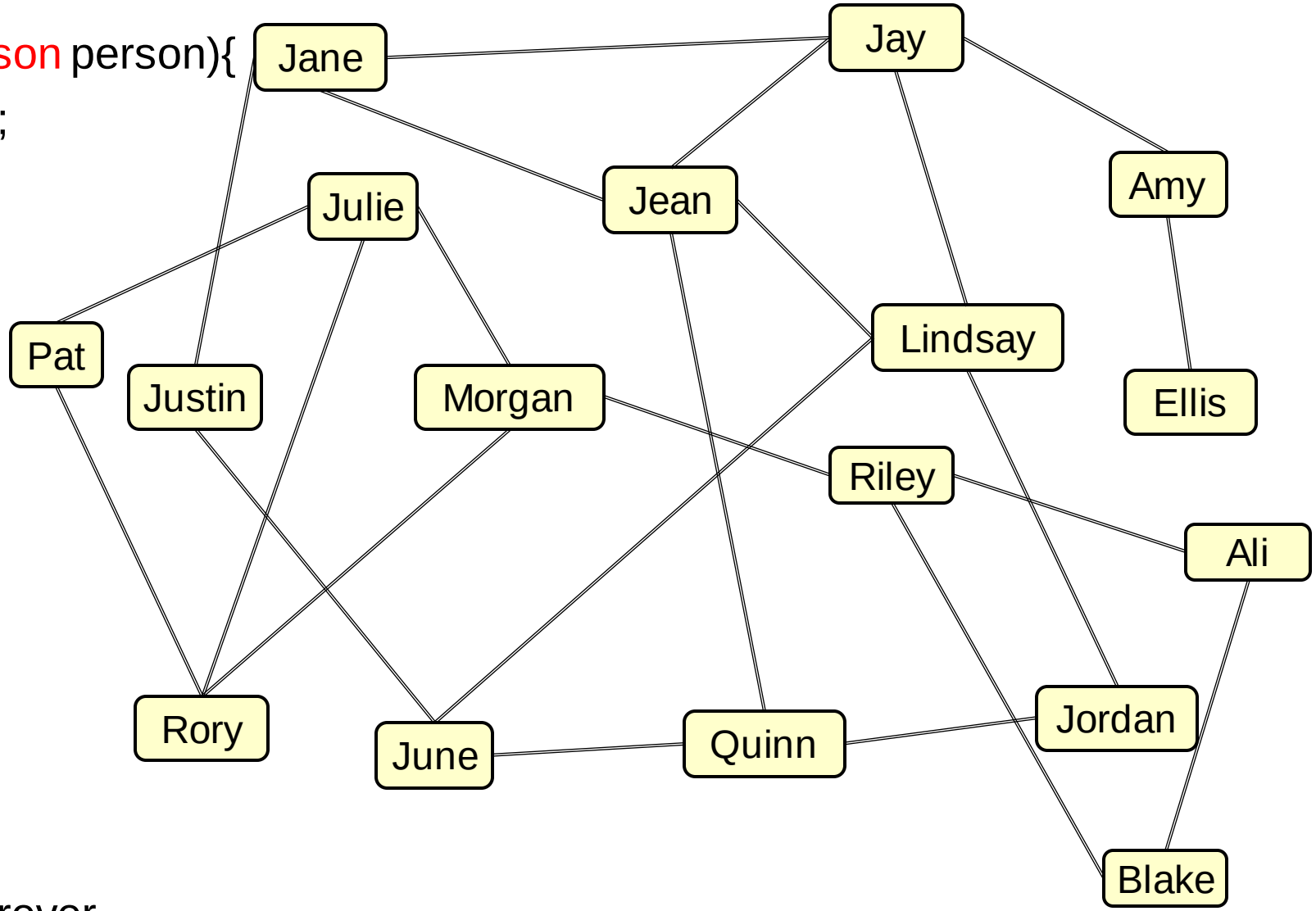


Graph Nodes.

```
public class SNPerson  implements Iterable<SNPerson>{
    private  String name;
    private Set<SNPerson> friends;

    public SNPerson(String nm){
        this.name = nm;
        this.firends = new HashSet<SNPerson>();
    }
    public String getName() { return name; }
    public void addFriend(SNPerson fr) {friends.add(fr); }
    public void removeFriend(SNPerson fr) {friends.remove(fr); }
    public boolean hasFriend(SNPerson fr) { return friends.contains(fr); }
    public Iterator<SNPerson> iterator() { return friends.iterator(); }
```

Traversing Graphs

```
/** Print all people in network of a Person  (Buggy) */
public void printNetwork(SNPerson person){
    UI.println(person.getName());
    for (Person friend : person){
        printNetwork(friend);
    }
}
```



Doesn't Work!!!!

The cycles mean we go round forever

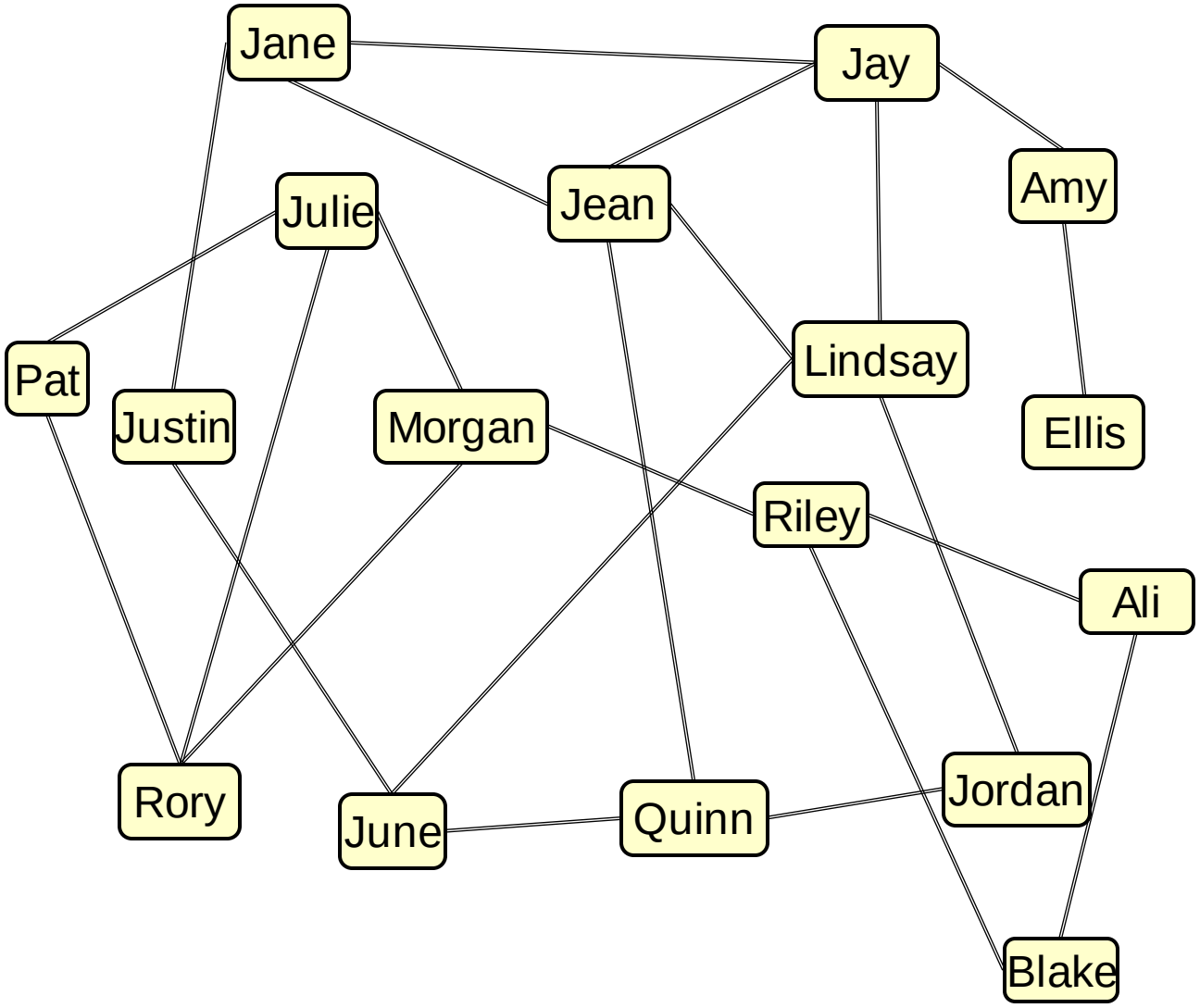
Traversing Graphs: marking nodes

Need to mark the nodes we have visited, and not re-visit them

```
/** Print all people in network of a Person */

public void printNetwork(SNPerson person){
    UI.println(person.getName());
    // mark person as visited
    for (Person friend : person){
        // if the friend is not visited, then
        printNetwork(friend);
    }
}
```

How do we mark nodes?

- Keep a Set of nodes we have visited, or
- Store a visited flag in the node

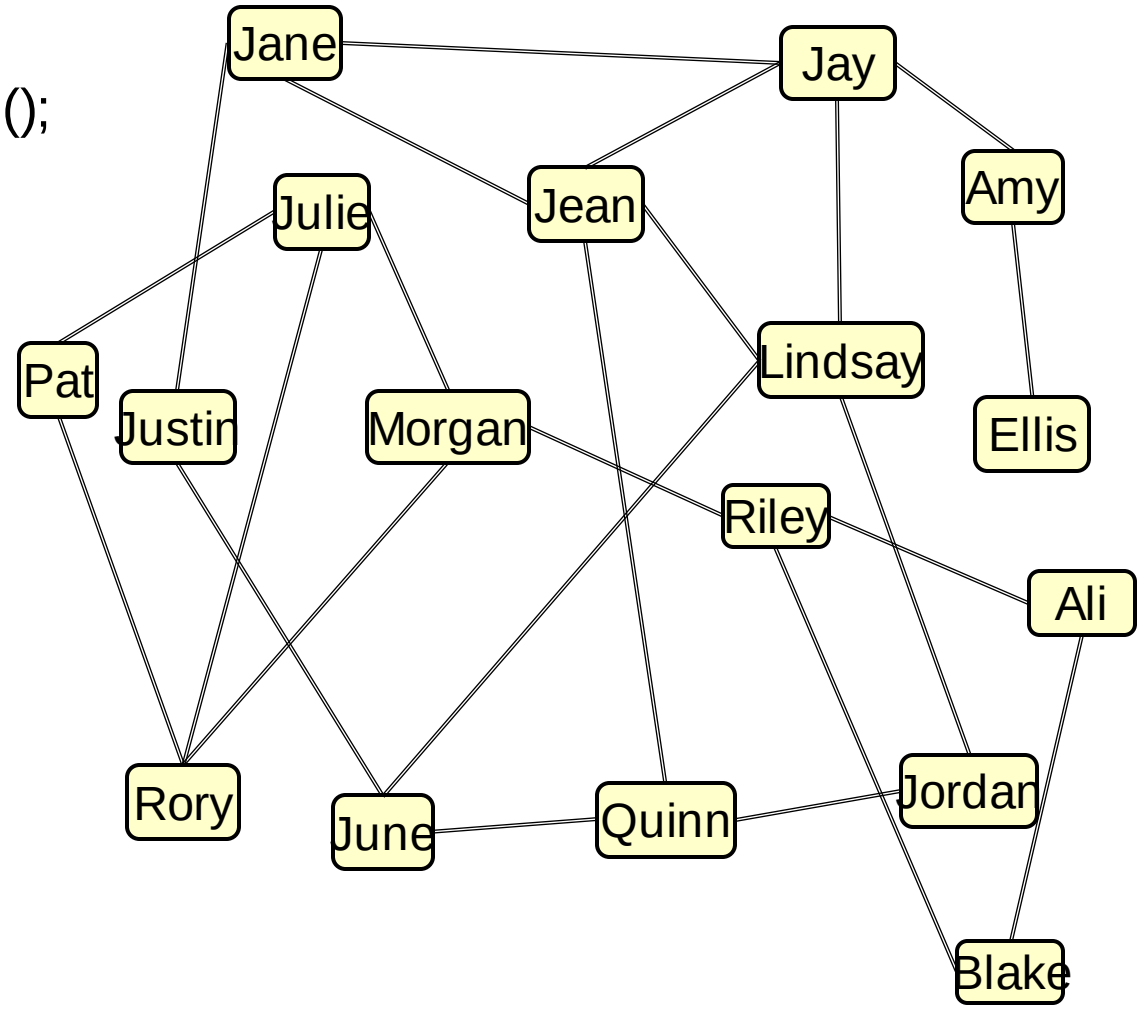


Traversing Graphs: Set of visited nodes

Keep Set of nodes we have visited

{ }

```
public void printNetwork(SNPerson person){
    printNetwork(person, new HashSet<SNPerson>();
}

public void printNetwork(SNPerson person,
                         Set<SNPerson> visited){
    UI.println(person.getName());
    visited.add(person);
    for (Person friend : person){
        if (! visited.contains(friend) ){
            printNetwork(friend, visited);
        }
    }
}
```

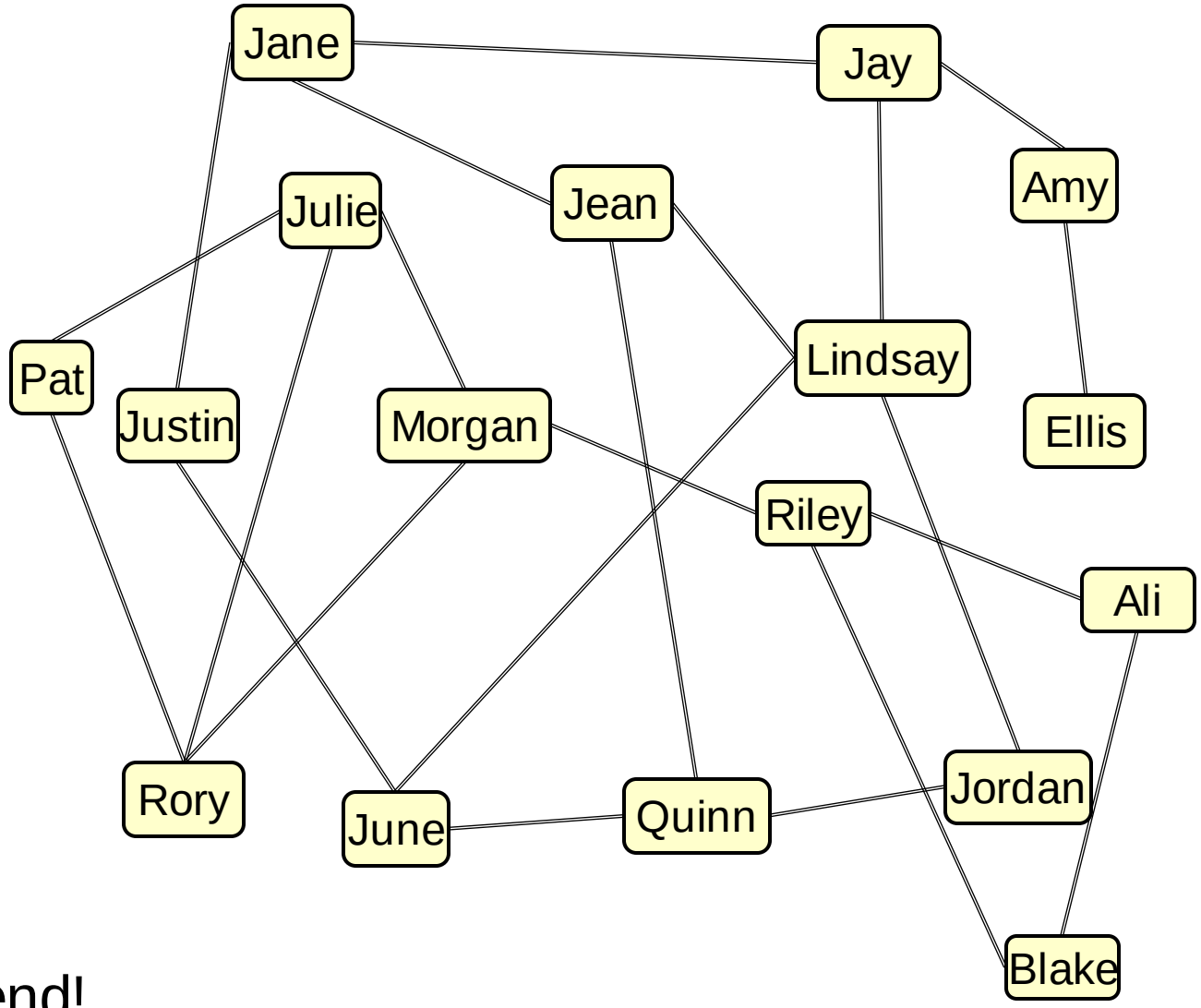


Still doesn't work if the graph is not connected!!

Traversing Graphs: visited flag inside node

Visited flag inside the node:

```
/** Print all people in network of a Person */

public void printNetwork(SNPerson person){
    UI.println(person.getName());
    person.visit();
     for (Person friend : person){
        if ( ! friend.isVisited()){
            printNetwork(friend);
        }
    }
}
```



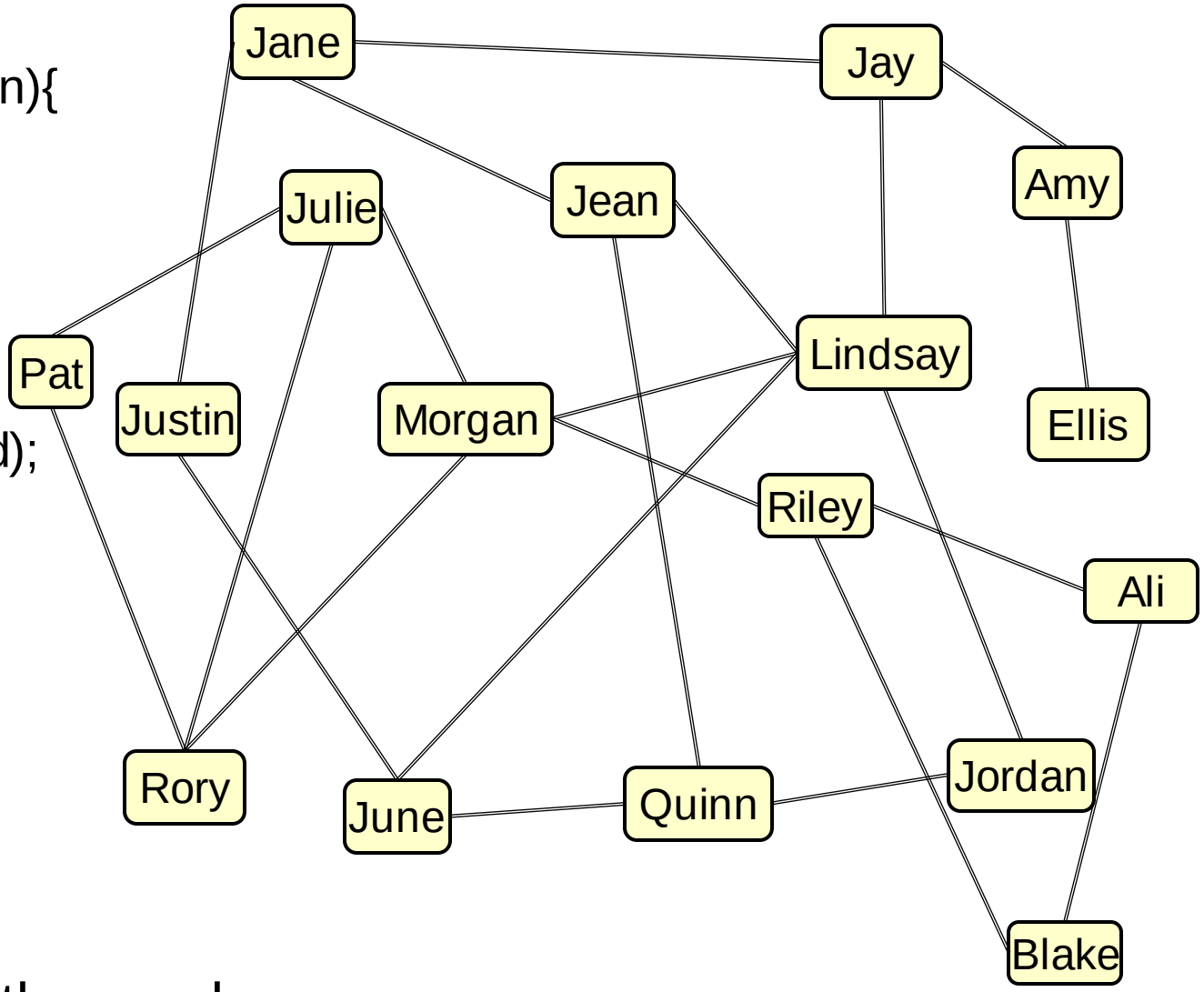
Need to reset all the visited flags at the end!

Graph Nodes (with visited flag)

```
public class SNPerson  implements Iterable<SNPerson>{
    private  String name;
    private Set<SNPerson> friends;
    private boolean visited;

    public SNPerson(String nm){
        this.name = nm;
        this.firends = new HashSet<SNPerson>();
    }
    public String getName() { return name; }
    public void addFriend(SNPerson fr)       { friends.add(fr); }
    public void removeFriend(SNPerson fr) { friends.remove(fr); }
    public boolean hasFriend(SNPerson fr) { return friends.contains(fr); }
    public Iterator<SNPerson> iterator()     { return friends.iterator(); }

    public void visit()     {visited=true; }
    public void unvisit() {visited=false; }
    public boolean isVisited() {return visited; }
```

Traversing Graphs : count connected nodes

```
/** Find number of friends in network */

public int countConnected(SNPerson person){
    person.visit();
    int count =1;
    for (Person friend : person){
        if ( ! friend.isVisited()) {
            count += countConnected(friend);
        }
    }
    return count;
}
```



Traversing a graph makes a tree within the graph.

Traversing Graphs: connectedTo

```
/** Are two people connected in the network */

public boolean connectedTo(SNPerson person, SNPerson query){
    if (person.equals(query) ) {
        return true;
    }
    person.visit();
    for (Person friend : person){
        if ( ! friend.isVisited()  &&  connectedTo(friend, query) ) {
            return true;
        }
    }
    return false;
}
```

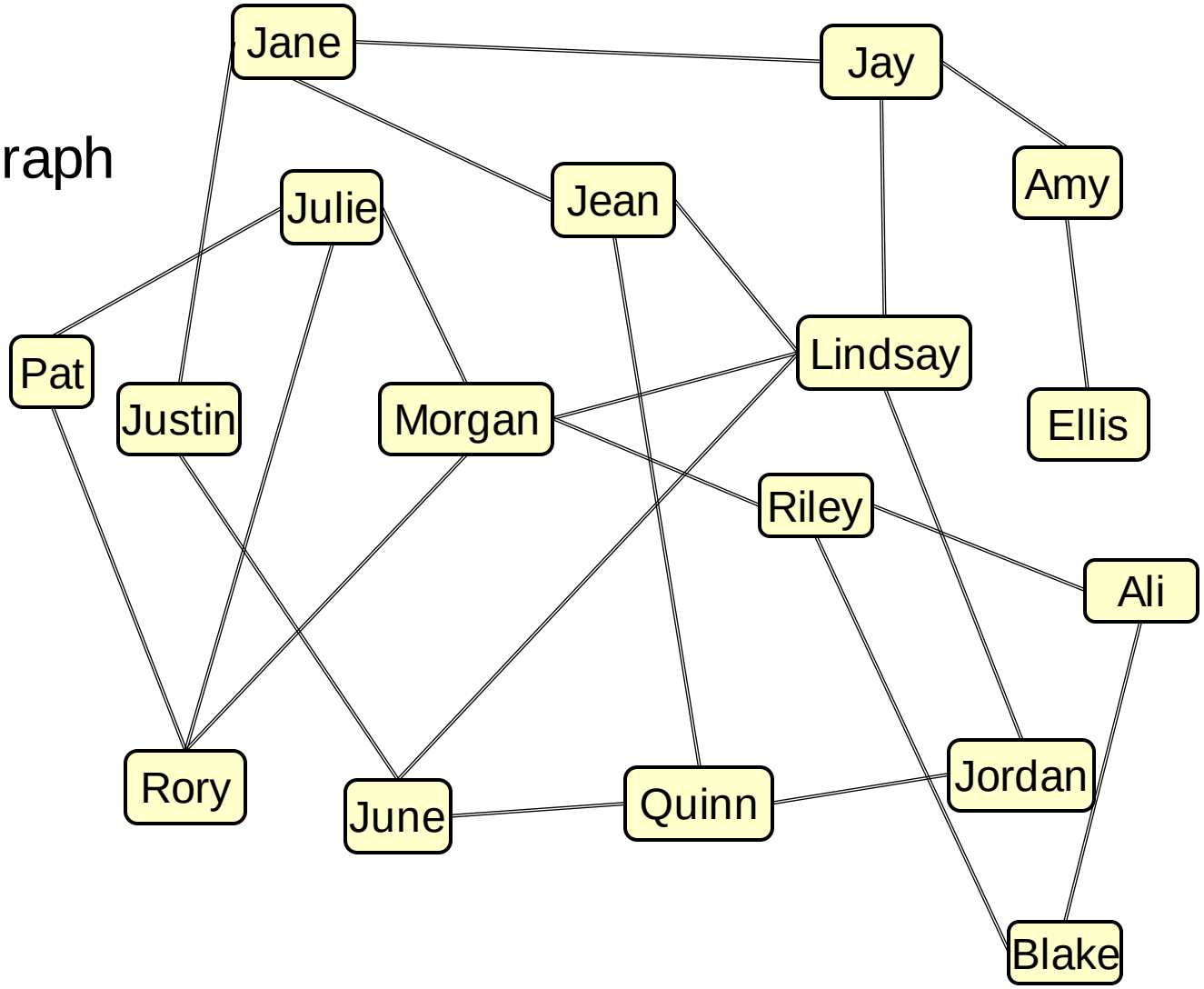
Traversing Graphs: iterative

```
/** Are two people connected in the network */

public boolean connectedTo(SNPerson person, SNPerson query){
    Stack<SNPerson> stack = new ArrayDeque<SNPerson>();
    Set<SNPerson> visited = new HashSet<SNPerson>();
    stack.push(person);
    while (! isEmpty()){
        SNPerson p = stack.pop();
        visited.add(p);
        if (p.equals(query) ) { return true; }
        for (Person friend : person){
            if ( ! friend.contains(friend)) { stack.push(friend); }
        }
    }
    return false;
}
```

Is Graph Connected?

- How do I represent this graph?
- Just having one node isn't enough!
- Need to store the set of nodes in the graph

```
private Set<SNPerson> graph;
```



Is it connected?

pick a node,

find all the connected nodes

does this cover all the nodes?