JASECI

Getting Started

Coursework Series : Week 2

Course Outline & Outcomes

Learn about the tools Jaseci offers to **ease the development** and Setting them up

Graphs, Nodes and Edges in Depth

Walkers, Actions in depth

Creating a small Non-AI Application with Jaseci (Step by Step)

Setting Up

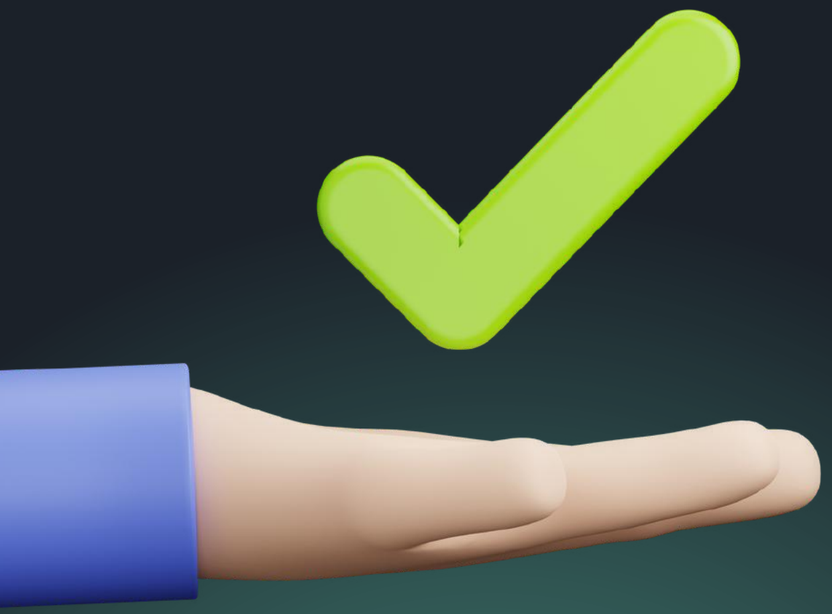
Installing the Tools Needed

- Installing Jaseci and Jaseci Server
  - **pip install jaseci jaseci-serv**
- Installing Jaseci Studio
  - **Go to https://github.com/Jaseci-Labs/jaseci/releases**
  - **Download the necessary package**
  - **Install the Package**

- Prerequisites
  - Python 3.8 or above
  - VSCode

JASECI

Setting Up the Environment



- Running the Jaseci Server
  - **jsserv makemigration base**
  - **jsserv makemigration**
  - **jsserv migrate**
  - **jsserv createsuperuser**
    - Enter your email and password (Rember this we are going to need it)
- Running Jaseci Studio
  - **Open Jaseci Studio and Enter your login info and jaseci server address and login**
- Connecting JSCTL with the Jaseci Server
  - **login HTTP://localhost:<port>**
  - Enter the login info

JASECI

THE APP

Explanation

The app you're building will let users load a family tree data file in JSON format, create a graph using Jaseci with each person as a node and relationships as edges, and visualize it in Jaseci Studio. Users can then use Jaseci's simulation capabilities to analyze the graph and explore family dynamics.

- The app can load JSON files (Actions)
- We will create nodes for each person (Nodes)
- Connect them to each other using edges (Edges)
- Visualize the Graph in Jaseci Studio (Graph)

What is a Node?

In Jaseci, **nodes** are a fundamental concept that represents entities in a graph. There are two types of nodes - root nodes and generic nodes. Root nodes are the starting points of a graph, while generic nodes are customizable and can be used throughout the application. Nodes can have abilities, which are self-contained compute operations that can interact with the context and local variables of the node.

Example Code of Node

```
node person{
    has name, age, birthday, profession;
}

walker init {
 person1 = spawn here node::person(name = "Josh", age = 32);
 person2 = spawn here node::person(name = "Jane", age = 30);

 std.out("Context for our people nodes");
 for i in -->{
    std.out(i.context);
 }
```

Nodes in General

```
node example_node {
    has name, count;

    can compute_sum {
        sum = 0;
        numbers = [1,2,3];
        for i in numbers{
            sum += i;
        }
        name = "Sum of first " + count.str + " numbers";
        report {"Computed sum: ": sum};
    }
}

walker init{
    spawn here ++>node::example_node;
    root{
        take-->[0];
    }
    example_node{
        here::compute_sum;
    }
}
```

Nodes with Abilities

JASECI

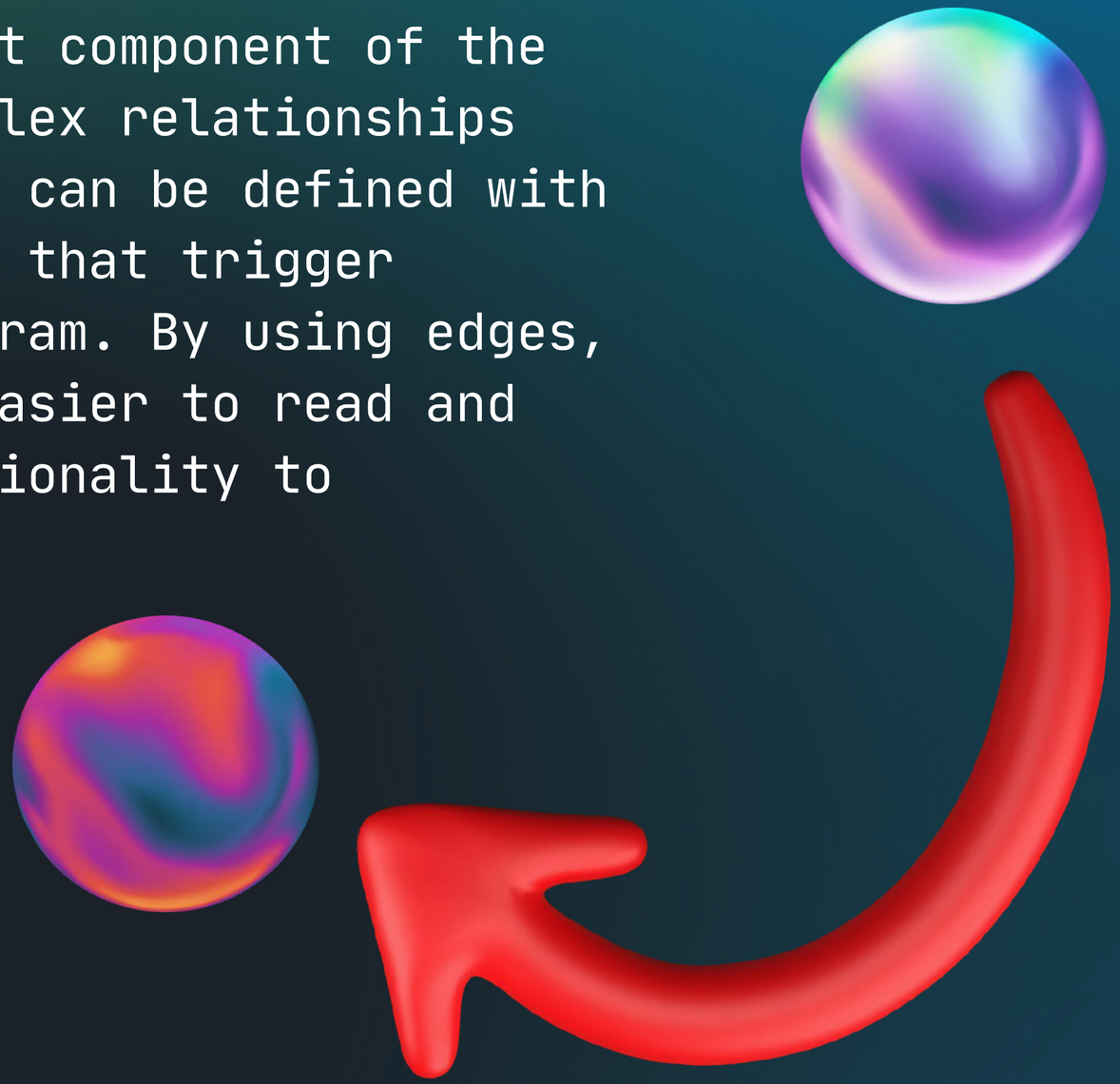
For our app we need only 2 Nodes

Person Node

```
node person {
    has id;
    has name;
    has date_of_birth;
    has profession;
    has gender;
    has deceased;
    has parents;
    has children;
    has spouse;
}
node family_root;
```

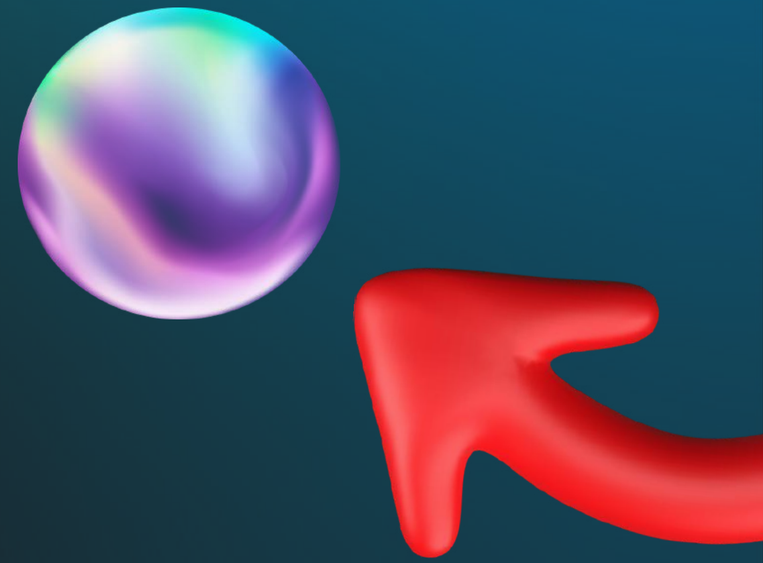
What is an Edge?

In Jaseci, **edges** are an important component of the graph structure that enable complex relationships between nodes. Custom edge types can be defined with specific behaviors or conditions that trigger actions or behaviors in the program. By using edges, code can be made more modular, easier to read and maintain, and can add more functionality to applications



Example Code for Edges

```
edge name_of_edge{
    has name_of_variable;
}
```



For our app we need 2 type of edges

Children Edge

Narried Edge

```
edge married;
edge children;
```

JASECI

When Nodes and Edges are combined is what makes a **Graph.**

In Our case, it is a **family tree**

What is a Walker?

Walkers in Jaseci are a unique abstraction for spacial problem solving. They allow for programmatic execution with state retention as they move from node to node in a graph. Walkers have two types of variables, context variables that retain state as they travel through the graph, and local variables that are reinitialized for each node-bound iteration. Walkers present a new way of thinking about programmatic execution and introduce data spacial problem solving. The take command is used for control flow for node-bound iterations, and additional keywords and semantics such as disengage, skip, and ignore are also introduced.

JASECI

Example Code of a walker

```
walker second_walker{
    std.out("This is from second walker \n");
}

walker init{
    std.out("This is from init walker");
    root{
        spawn here walker::second_walker;
    }
}
```

```
This is from init walker
This is from second walker
```

```
#init walker traversing
walker init {
    root {
        start = spawn here ++> graph::example;
        take-->;
        }
    plain {
        ## Skipping the nodes with even numbers
        if(here.number % 2==0): skip;
        std.out(here.number);
        take-->;
    }
}
```

```
1
5
7
```

JASECI

Lets Run the code and See what is going on

Next week

1. Use STD actions to load a JSON file that contains the family members
2. Use abilities to perform certain tasks