BLOCKLY

To Create New Block

Step1-

Create Block-
```
const moveBlock = {
  "type": "move_block",
  "message0": "Move %1 %2 steps",
  "args0": [
    {
      "type": "field_dropdown",
      "name": "DIRECTION",
      "options": [
        ["Forward", "FORWARD"],
        ["Backward", "BACKWARD"]
      ]
    },
    {
      "type": "field_number",
      "name": "STEPS",
      "value": 1,
      "min": 1
    }
  ],
  "previousStatement": null,
  "nextStatement": null,
  "colour": 160,
  "tooltip": "Move forward or backward by a certain number of steps"
};
```

1. **Type-** `"type"` (mandatory): Specifies the type of the block. It is used to identify and differentiate this block from others in the Blockly workspace.
   The field values are user-defined values(**Like id of each block**)

1. **Message-** `"message0"` (mandatory): Represents the main text displayed on the block. In this case, it is set to "Move %1 %2 steps". The `%1` and `%2` are placeholders for the dropdown and number fields, respectively.

1. **Args[0]-** `"args0"` (mandatory): An array of argument objects that define the fields within the block. In this case, it contains two field objects.

1. `"previousStatement"` (optional): Specifies whether the block can be connected to a previous statement block. In this case, it is set to `null`, meaning this block does not have a previous statement connection.
2. `"nextStatement"` (optional): Specifies whether the block can be connected to a next statement block. Similar to `"previousStatement"`, it is set to `null` in this case.

“By setting the "previousStatement" or "nextStatement" property to null, as in the example given, it means that the block does not have the ability to be connected in that specific way. In the case of "move_block" or "turn_block" blocks, they are designed to be standalone actions without any specific requirement for a previous or next block in the sequence.

So, to summarize, "previousStatement" and "nextStatement" properties determine the connectivity options for a block, allowing it to be connected as the previous or next action in a sequence of blocks.**”**

“**means agar null nahi hoga to hme connect karna hi padega otherwise error hoga**”

1. **Color-**`"colour": 160` (optional): Sets the color of the block. It uses a numerical value that represents a specific color in the Blockly color palette. Here, it is set to 160, which corresponds to a shade of blue.

1. **Toolkit-**`"tooltip"` (optional): Provides a tooltip text that appears when hovering over the block. It gives a brief explanation of what the block does. In this case, it explains that the block allows movement in a specific direction by a certain number of steps.

In summary, the "type", "message0", "args0", and "name" fields are mandatory and necessary for defining the structure of the block. The dropdown and number fields within "args0" are also mandatory and allow users to input specific values. The "value", "min", "previousStatement", "nextStatement", "colour", and "tooltip" fields are optional and can be customized based on your specific requirements for the block.

Step2 – Register above block with blockly

```
Blockly.Blocks['move_block'] = {
  init: function() {
    this.jsonInit(moveBlock);
    this.setStyle('motion_blocks');
  }
};
```

By assigning the object with the init function to Blockly.Blocks['move_block'], you are effectively registering the block type and providing the initialization logic for the block. The init function will be called whenever a new instance of the 'move_block' block is created, allowing you to configure its behavior, structure, fields, and other properties

1. `Blockly.Blocks['move_block']`: This line registers a new block type with the name `'move_block'` in the `Blockly.Blocks` object. This allows Blockly to recognize and handle blocks of this type.

1. `init: function() { ... }`: This is the initialization function for the block. It is called when a new instance of the block is created.

The `jsonInit()` method is used to initialize a block instance by providing it with a JSON (object/block which we define above) definition. This JSON definition specifies the structure, fields, and other properties of the block.

When you call `jsonInit(moveBlock)`, it means you are initializing the current block instance (`this`) using the JSON definition stored in the `moveBlock` object.

Here's what happens when `jsonInit()` is called:

1. The block's structure is set up: The JSON definition contains information about the message displayed on the block, the arguments or fields it has, and other

structural details. `jsonInit()` uses this information to configure the block's appearance and layout.

2. The block's fields are set up: The JSON definition specifies the type and properties of each field in the block, such as dropdown menus, text inputs, or number inputs. `jsonInit()` uses this information to create the fields and associate them with the block instance.
3. Other properties are set up: The JSON definition may include additional properties like colors, tooltips, or connection details. `jsonInit()` takes care of setting up these properties for the block.

1. `this.setStyle('motion_blocks')`: The `setStyle()` method is used to set the visual style of the block. In this case, it sets the style to `'motion_blocks'`. The specific visual appearance and color scheme associated with the `'motion_blocks'` style would be defined elsewhere in the Blockly configuration.

<u>Step3- Generate JS Code</u>

```
javascriptGenerator['move_block'] = function(block) {
  const direction = block.getFieldValue('DIRECTION');
  const steps = block.getFieldValue('STEPS');

  let moveCode;
  if (direction === 'FORWARD') {
    moveCode = `moveForward(${steps});\n`;
  } else {
    moveCode = `moveBackward(${steps});\n`;
  }

  return moveCode;
};
```

When this function calls? – On Button Click this function calls

```
 const generateCode = () => {
       var code = javascriptGenerator.workspaceToCode(
         primaryWorkspace.current
       );
       console.log(code);
   }
```

var code = javascriptGenerator.workspaceToCode(primaryWorkspace.current);: This line calls the workspaceToCode() function provided by javascriptGenerator, passing the current Blockly workspace as an argument (primaryWorkspace.current). The workspaceToCode() function generates the code for the entire Blockly workspace and returns it as a string.

- When you use Blockly to generate code(like above on click of button generateCode() calls), you typically invoke a code generation method or trigger an event that initiates the code generation process. **During this process, Blockly traverses the blocks in your workspace and determines their types.**

When Blockly encounters a block of a specific type for which a code generator function has been defined(genrater.js file), it calls that code generator function, passing the corresponding block instance as an argument(and that block we receive inside function parameter like `javascriptGenerator['move_block'] = function(block){}`**). This allows the code generator function to extract the necessary information from the block(like- const direction , const steps information) and generate the appropriate code.**