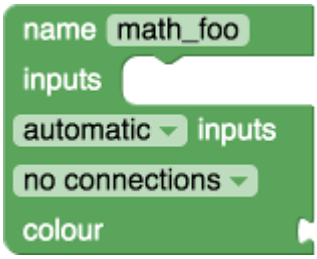
自定义块：块范例

设计使用Blockly的应用程序时，有几种范例可供选择。这些选择应尽早考虑，因为它们会影响用户所需的功能块。

构造

许多Blockly应用程序用于描述配置，而不是可执行程序。配置应用程序通常从初始化工作空间上的一个根级别块开始。一个很好的例子是Blockly Developer Tools的Block Factory选项卡：




```
var xml = '<xml><block type="factory_base" deletable="false" movable="false">
</block></xml>';
Blockly.Xml.domToWorkspace(Blockly.Xml.textToDom(xml), workspace);
```

这将创建一个不可删除的，不可移动的块，其中包含所有用户的配置。可以随时将工作空间导出到XML，以确定当前配置。

此类应用程序可能希望自动禁用任何未连接到根块的块。只需一行即可在Blockly的Web版本上完成此操作：

```
workspace.addChangeListener(Blockly.Events.disableOrphans);
```

串行程序

大多数Blockly应用程序旨在创建串行程序。用户将按顺序执行的块堆叠在一起。

工作区中的每个（非禁用）块都将构成程序的一部分。如果有多个堆栈块，则首先执行较高的堆栈。（如果两个堆栈的高度大致相同，则优先考虑左侧（在RTL模式下为右侧）的堆栈。）

工作空间可以随时导出为可执行代码。此代码可以在JavaScript的客户端（使用eval或JS Interpreter）执行，也可以在服务器端以任何语言执行。

```
var code = Blockly.JavaScript.workspaceToCode(workspace);
```

并行程序

一些Blockly应用程序选择并行而非串行执行所有块堆栈。一个例子是音乐应用程序，其中鼓循环与旋律同时运行。

实现并行执行的一种方法是使用无头工作空间生成多个代码段：

```
var xml = Blockly.Xml.workspaceToDom(workspace);
// Find and remove all top blocks.
var topBlocks = [];
for (var i = xml.childNodes.length - 1, node; block = xml.childNodes[i]; i--) {
  if (block.tagName == 'BLOCK') {
    xml.removeChild(block);
    topBlocks.unshift(block);
  }
}
// Add each top block one by one and generate code.
var allCode = [];
for (var i = 0, block; block = topBlocks[i]; i++) {
  var headless = new Blockly.Workspace();
  xml.appendChild(block);
  Blockly.Xml.domToWorkspace(xml, headless);
  allCode.push(Blockly.JavaScript.workspaceToCode(headless));
  headless.dispose();
  xml.removeChild(block);
}
```

如果目标语言是JavaScript，则可以使用该数组创建多个JS解释器以同时执行。如果目标语言是类似Python的语言，则可以将数组组装为使用线程模块的单个程序。

与任何并行程序一样，必须对任何共享资源（例如变量和函数）做出谨慎的决定。

事件驱动程序

事件处理程序只是由系统而不是由程序调用的函数。这些块可以封装要执行的块堆栈，也可以是位于块堆栈顶部的标头。

一些开发人员喜欢在事件块的顶部添加一个“帽子”，以使它们看起来与其他块不同。这不是Blockly的默认外观，但可以通过设置Blockly.BlockSvg.START_HAT = true;或添加主题并在block style上设置hat选项来添加。有关在主题上设置帽子的更多信息，请参见 此处。

在事件驱动的模型中，也可以为程序启动创建处理程序。在此模型下，工作空间上未连接到事件处理程序的任何块都将被忽略并且不会执行。

设计使用事件的系统时，请考虑是否可能或希望支持同一事件处理程序的多个实例。