KookaBlockly Reference Guide

Release v1.10.0

Julian Dinsdale and Tony Strasser

2024-01-29

CONTENTS

1 Part 1 - Working With KookaBlockly **3**

- 1.1 Introduction to KookaBlockly . . . 3
- 1.2 Installing KookaBlockly . . . 5
- 1.3 Using the KookaBlockly Application . . . 15
- 1.4 KookaBlockly Conventions . . . 25

2 Part 2 - KookaBlockly Function Blocks Reference **29**

- 2.1 Control . . . 29
- 2.2 Clock . . . 31
- 2.3 Display . . . 36
- 2.4 Buttons . . . 43
- 2.5 LEDs . . . 45
- 2.6 Pins . . . 47
- 2.7 Sensors . . . 53
- 2.8 Actuators . . . 63
- 2.9 Radio . . . 66
- 2.10 Logging . . . 71
- 2.11 Boolean . . . 73
- 2.12 If–Else . . . 76
- 2.13 Loops . . . 78
- 2.14 Strings . . . 81
- 2.15 Lists . . . 83
- 2.16 Math . . . 91
- 2.17 Variables . . . 100
- 2.18 Functions . . . 103
- 2.19 Advanced . . . 108

3 Glossary of Terms **113**

Index **117**

KookaBlockly is a powerful standalone visual editor designed for creating program scripts for **Kookaberry** and related microprocessors. This editor operates on a drag-and-drop interface, making it beginner-friendly and highly intuitive.

This document describes how to use to the **KookaBlockly** visual scripting tool.

KookaBlockly is part of the **KookaSuite** script editing toolset which was commissioned by the AustSTEM Foundation and created by Damien George for the **Kookaberry**.

This guide is for **KookaBlockly** v1.10.0.

The document is in TWO parts:

1. Working with **KookaBlockly** - relates to **KookaBlockly** set-up, basic screen displays and usage.
2. A Reference Document for the visual functional blocks in **KookaBlockly**.

CHAPTER

ONE

PART 1 - WORKING WITH KOOKABLOCKLY

In this Part 1 of the **KookaBlockly Reference Guide**, **KookaBlockly** in introduced, then instructions are given on how the **KookaSuite** software package, is installed on a personal computer, how **KookaBlockly** is used, and finally the conventions used by **KookaBlockly** are explained.

1.1 Introduction to KookaBlockly

1.1.1 KookaBlockly: Visual Programming Editor for Kookaberry Microprocessor Boards

KookaBlockly is a powerful standalone visual editor designed for creating program scripts for **Kookaberry** and related microprocessor boards. This editor operates on a drag-and-drop interface, making it beginner-friendly and highly intuitive. It's built upon the open-source Google Blockly library (Apache 2 license), created by Google to facilitate the development of beginner-friendly programming languages.

Fig. 1.1 shows a **KookaBlockly** script assembled from visual function blocks dragged onto the workspace from the palette of blocks on the left of the display. The blocks click together like pieces of a jigsaw puzzle to form a series of steps that the **Kookaberry** microcomputer will perform.

The example shown above shows a loop that writes a welcome message on the **Kookaberry** display and flashes the **Kookaberry**'s **LEDs**. It then sleeps for 2 seconds and then goes back to the beginning of the loop. The loop will run until the **Kookaberry** is reset or power is removed.

KookaBlockly was created by Damien George (George Robotics – MicroPython) in collaboration with Kookaberry Pty Ltd. It also received support from the AustSTEM Foundation, the Warren Centre, and the Vonwiller Foundation.

1.1.2 Key Features

Intuitive Visual Interface:

Users can create syntactically correct scripts and programs effortlessly, even without prior knowledge of any programming language.

KookaBlockly enables users to assemble visual blocks into structured **MicroPython** (Python 3.0) code.

Compatibility:

The generated code can be utilized on most microprocessor boards that use **MicroPython**, but is particularly suited to those with **Kookaberry** firmware for **STM** and **RP2040** microprocessors.

Platform Compatibility:

KookaBlockly runs as a standalone program on personal computers with **Microsoft Windows** 10 or 11, **Apple MacOS**, or **Raspberry Pi Raspbian** operating systems.

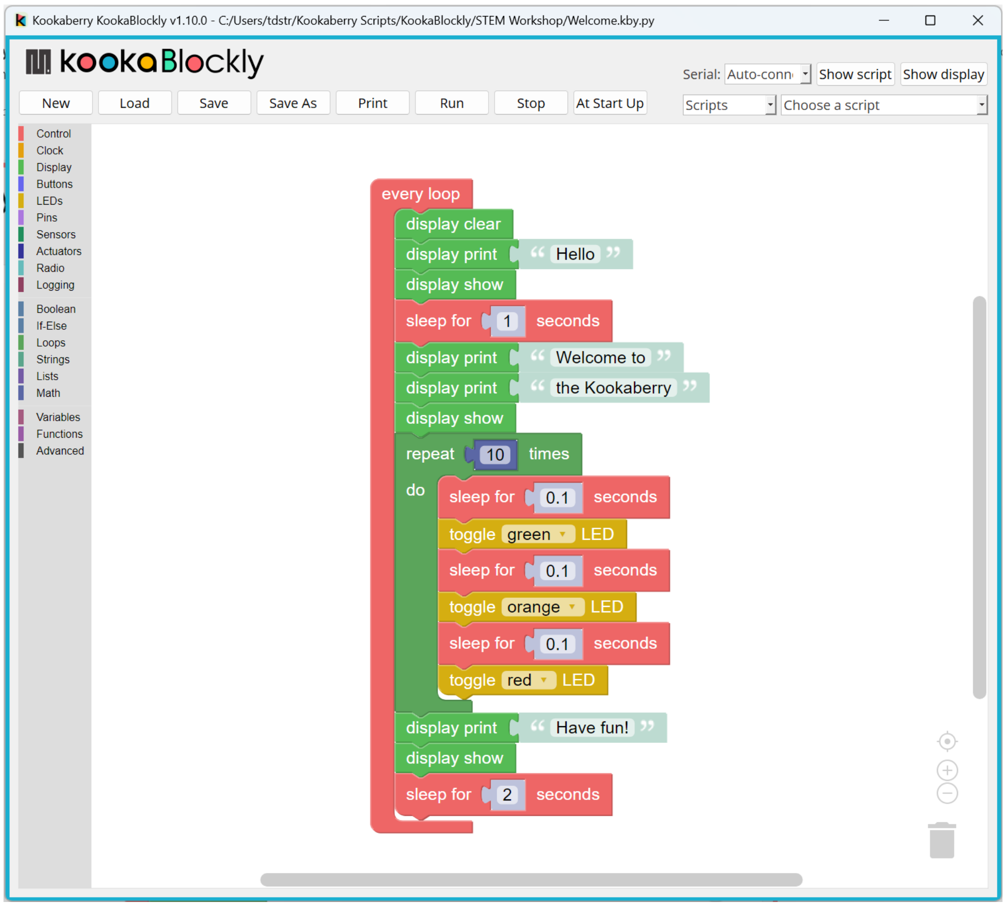


Fig. 1.1: This is the **KookaBlockly** display with an example **KookaBlockly** script.

Easy Access:

The latest version of **KookaBlockly** can be conveniently downloaded from the **Kookaberry GitHub** repository at https://github.com/kookaberry/kooka-releases/releases.

Follow the *Installing KookaBlockly* guide in the next section to install **KookaBlockly**.

1.1.3 Programming With KookaBlockly

Using **KookaBlockly** is straightforward and enjoyable.

Users can drag and drop visual code blocks into the workspace, where they can be seamlessly interlocked or snapped together using sockets.

These sockets represent fundamental code concepts, including program controls (activation, termination, loops, and decisions), actions, and result computations (variables, values, mathematical and logical expressions).

The intuitive visual process empowers users to apply programming concepts and principles when designing scripts or programs, eliminating the need to worry about the syntax and semantics of MicroPython.

With **KookaBlockly**, programming becomes an enjoyable and accessible endeavour.

1.1.4 AustSTEM Learning Hub

AustSTEM has assembled a collection of resources on its Learning Hub at https://learn.auststem.com.au. These resources complement the material in this manual with examples, lesson plans, descriptions of equipment and of their application.

1.2 Installing KookaBlockly

KookaBlockly is part of the **KookaSuite** set of code development and editing tools for the Kookaberry microcomputer and other microcomputer boards that can use the Kooka firmware.

The tools that are in **KookaSuite** are:

KookaBlockly

a powerful standalone visual editor designed for creating program scripts.

KookaIDE

a text editor for creating and editing MicroPython program scripts and directly interacting with the Kookaberry control console.

IDE is short for Integrated Development Environment.

KookaTW

A virtual Kookaberry user interface that replicates the physical user interface on a Kookaberry and provides a user interface for compatible microprocessor boards that do not have a physical user interface.

TW originated as Teacher's Window, but also stands for TWin, or in some cases Training Window.

1.2.1 Downloading KookaSuite

The latest version of **KookaBlockly** can be conveniently downloaded from the Kookaberry **GitHub** repository at https://github.com/kookaberry/kooka-releases/releases.

Choose the latest version compatible with your personal computer. **KookaSuite** versions available are for:

- Microsoft Windows V10 and later
- Apple MacOS V10.15 and later
- Raspberry Pi OS (32 bit Debian v12 [bookworm])

Click on the hyperlink for the appropriate version of **KookaSuite** and download it to a folder (default is in the **Downloads** folder) on your personal computer.

1.2.2 Installing KookaSuite on Microsoft Windows

1. Double-click on the downloaded `KookaSuite-<version>-Win64.msi` file to launch the Windows Installer. The display in Fig. 1.2 will then appear.

Fig. 1.2: Click on **Next** to proceed.

1. **KookaSuite** does not (as yet) have an application trust certificate, so Windows Defender will alert you with the dialogues in Fig. 1.3 and Fig. 1.4.

1. The installer will then show the **Kookaberry Licence Agreement**. The agreement contains a liability disclaimer, then a series of open-source licences for the software that is embedded within **KookaSuite**.

   To obtain a printed copy of the licence, press **Print**.

   Please read the licence conditions and if you accept them, click on the acceptance checkbox to place a tick (as shown in Fig. 1.5) and then click on **Next**.

4. The dialogue in Fig. 1.6 will then appear showing where on your computer the **KookaSuite** programs will be installed.

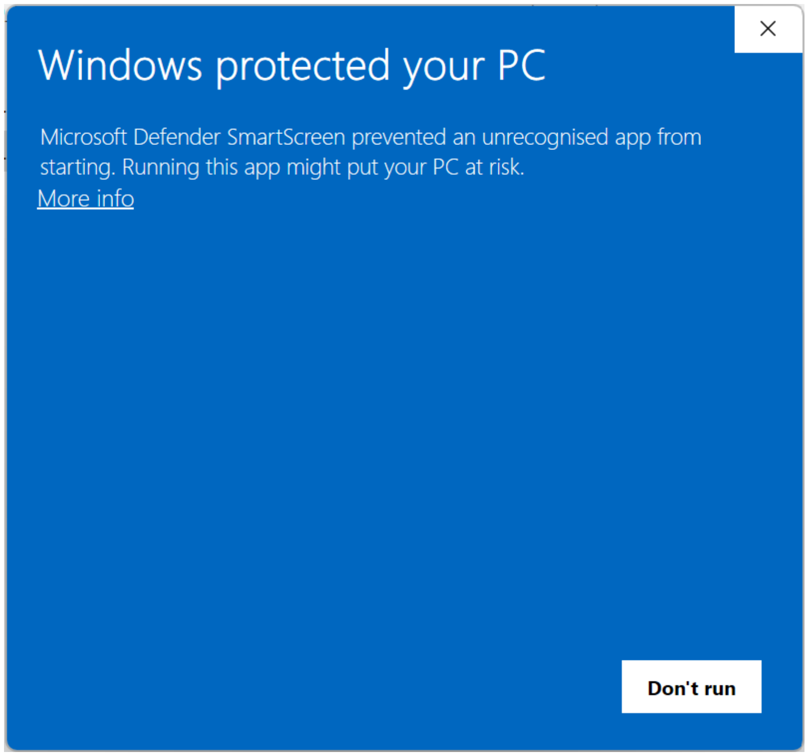


Fig. 1.3: Click on **More info** to proceed to the next dialogue.

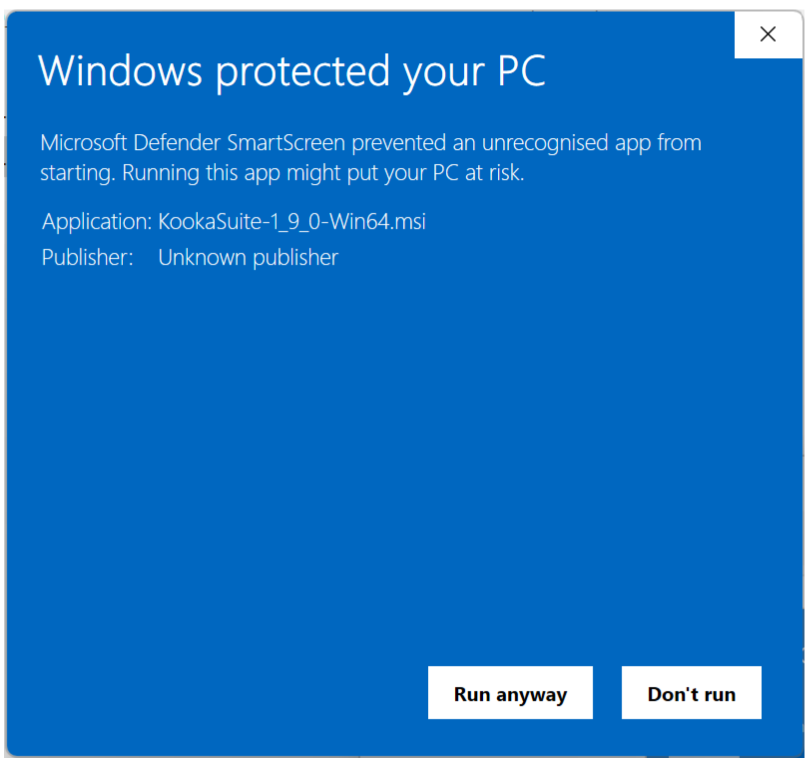


Fig. 1.4: Click on **Run Anyway** to proceed.

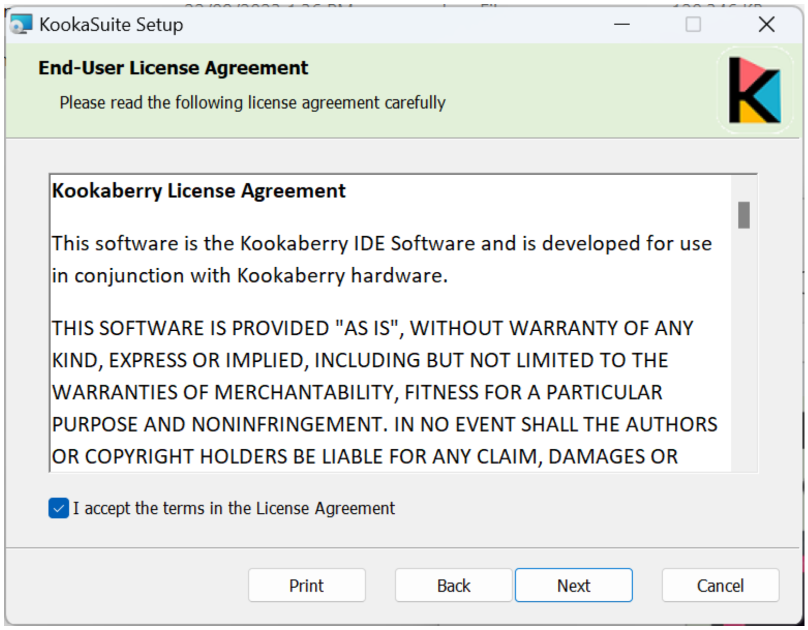


Fig. 1.5: Click the checkbox to accept the licence, then click on **Next** to proceed.

Usually the default location of `C:\Program Files\Kookaberry\KookaSuite` is fine, but you or your system administrator may wish to put them elsewhere. If so, click on **Change** and select the prefered location using the file explorer dialogue which will open.

5. The next dialogue, shown in Fig. 1.7, specifies the folder in which **KookaSuite** will store files.

   The default location is `C:\Users\Public\Kookaberry Scripts\` which all users share on a Windows PC. If another location (for example) `C:\Users\<your account>\Kookaberry Scripts\` which is unique and private to <your account>) is desired, click on **Change** and select the preferred location using the file explorer dialogue which will open.

6. A dialogue then appears, shown in Fig. 1.8, that provides the opportunity to select which elements if not all of **KookaSuite** are to be installed. It is recommended that all elements be installed for a fully functional **KookaSuite**.

7. A dialogue with a progress bar that tracks the installation progress will appear as in Fig. 1.9.

   There may be a Windows alert asking for permission to proceed. Accept the installation by clicking **Yes**.

   The progress bar will then continue and when it reaches completion the Completed dialogue will appear.

1.2.3 Installing KookaSuite on MacOS

1. Double-click on the downloaded `KookaSuite-<version>-macOS.dmg` file to open it. You will see it contains the three **KookaSuite** apps, as in Fig. 1.10.

2. Create a suitably named folder in the Macintosh `Applications\` folder and drag the **KookaSuite** apps into it, as shown in Fig. 1.11.

   KookaBlockly will then be available to launch (as will **KookaIDE** and **KookTW**) from the Applications icon in the Macintosh taskbar and by any other regular methods for starting Macintosh applications.

If a **KookaSuite** tool has not been run on the Macintosh before, a security warning notice may come up. The procedure for running any **KookaSuite** tool for the first time is given by the Apple Support website here: https://support.apple.

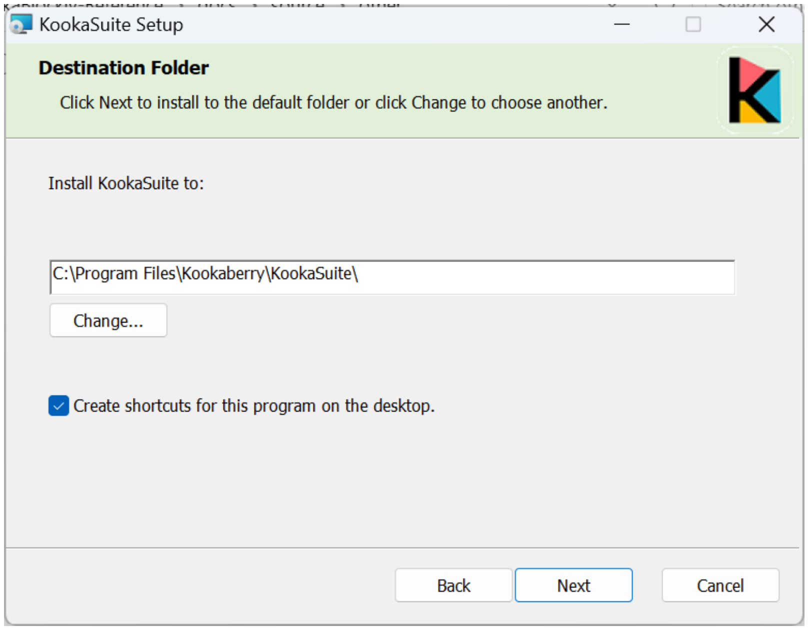


Fig. 1.6: Installation location dialogue. Click on **Next** to proceed.

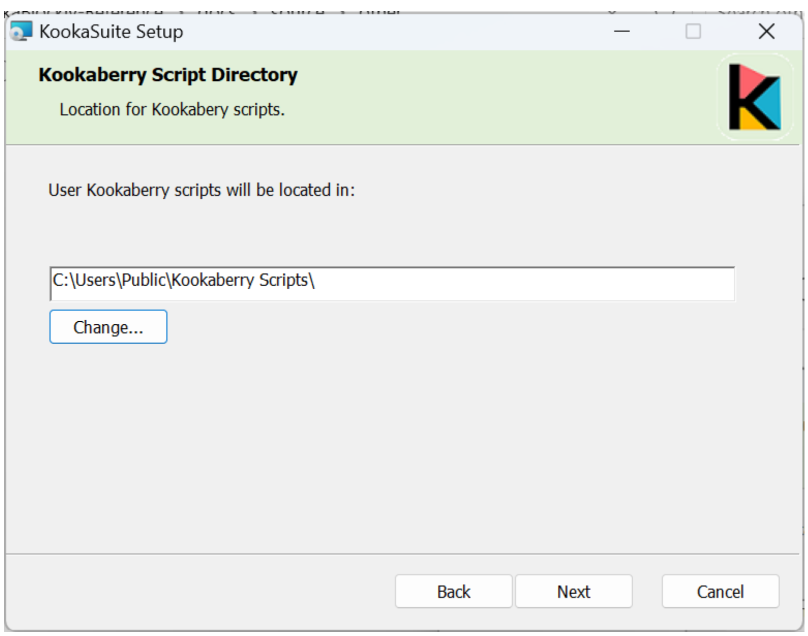


Fig. 1.7: Scripts location dialogue. Click **Next** to proceed.

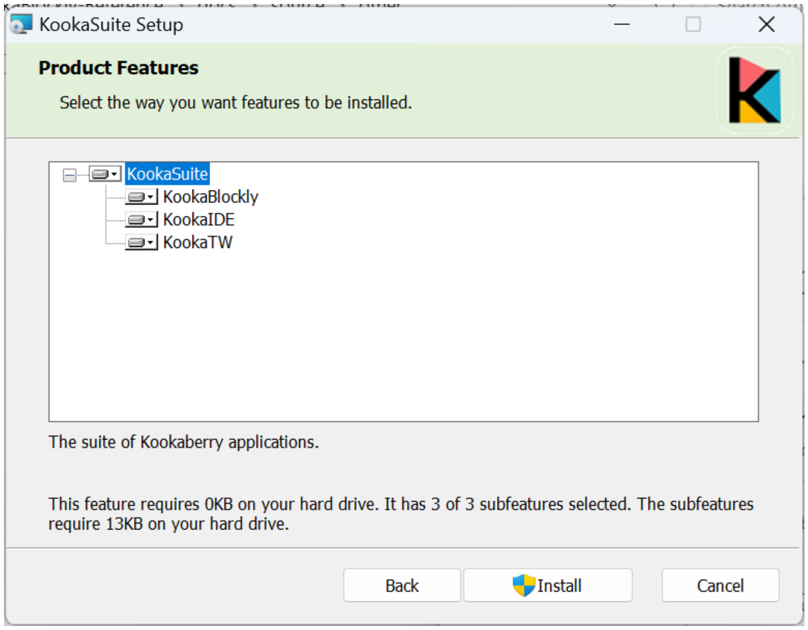


Fig. 1.8: Press **Install** to proceed with the **KookaSuite** installation.

Fig. 1.9: Click on **Finish** to exit the Windows Installer.

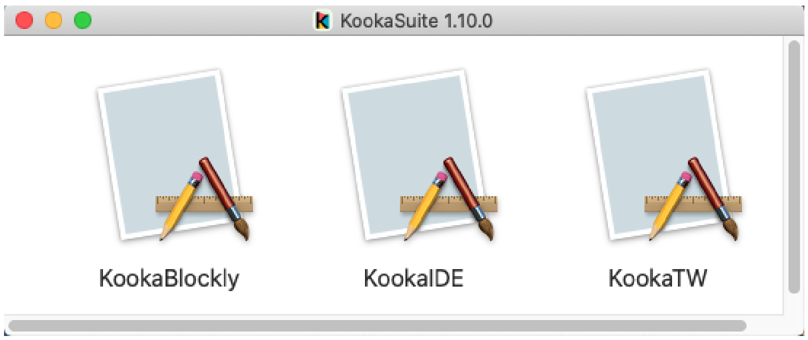


Fig. 1.10: The contents of the MacOS **KookaSuite** download package.

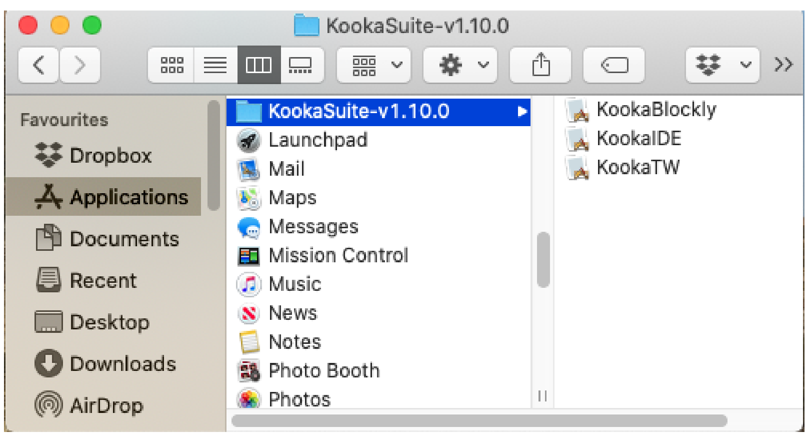


Fig. 1.11: **KookaSuite** apps copied to the Applications folder.

com/en-us/HT202491. After that the Macintosh will trust the software and allow it to run.

1.2.4 Installing KookaSuite on Raspberry Pi

KookaSuite has been compiled to run on the 32 bit version of the Raspberry Pi OS (Operating System), which is based on Debian Linux v12, known as “bookworm”. **KookaSuite** will not run on earlier versions of the Raspberry Pi OS, nor on the 64 bit version (unless you install dual architecture libraries, which can be complicated).

If your Raspberry Pi OS is an earlier version, you will need to update it. First back-up your Raspberry Pi on some removable media e.g. a USB memory stick. The easiest way is to flash the current 32 bit version onto a new SD-card following the instructions here: https://www.raspberrypi.com/software/ This will set up a new Raspberry Pi OS without any of your files on it. Retain the old Raspberry Pi SD card in case you need to retrieve some information from the older operating system. Then restore your data backup data into the home folder of the new Raspberry Pi OS.

Then proceed to download the `KookaSuite-<version>-RPi.zip` file from the the Kookaberry **GitHub** repository at https://github.com/kookaberry/kooka-releases/releases.

Unzip the downloaded file into the home folder. This will create a folder containing the three executables **KookaBlockly**, **KookaIDE** and **KookaTW** as shown in Fig. 1.12.

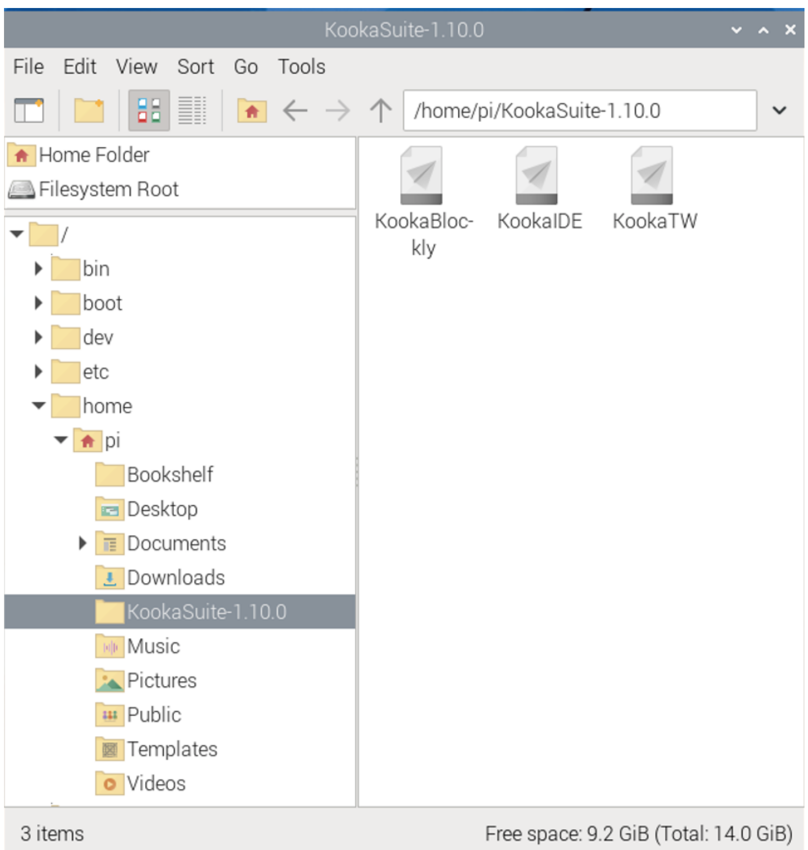


Fig. 1.12: **KookaSuite** apps copied to a folder in the **Raspberry Pi**’s home folder.

Using the terminal program, install the needed Qt5 modules:

Listing 1.1: Installing QT5

```
sudo apt install libqt5webkit5
sudo apt install libqt5websockets5
sudo apt install libqt5serialport5
```

If desired, create Raspberry Pi menu items under `Programming` using the `Preferences/Main Menu Editor` as shown in Fig. 1.13 and Fig. 1.14.

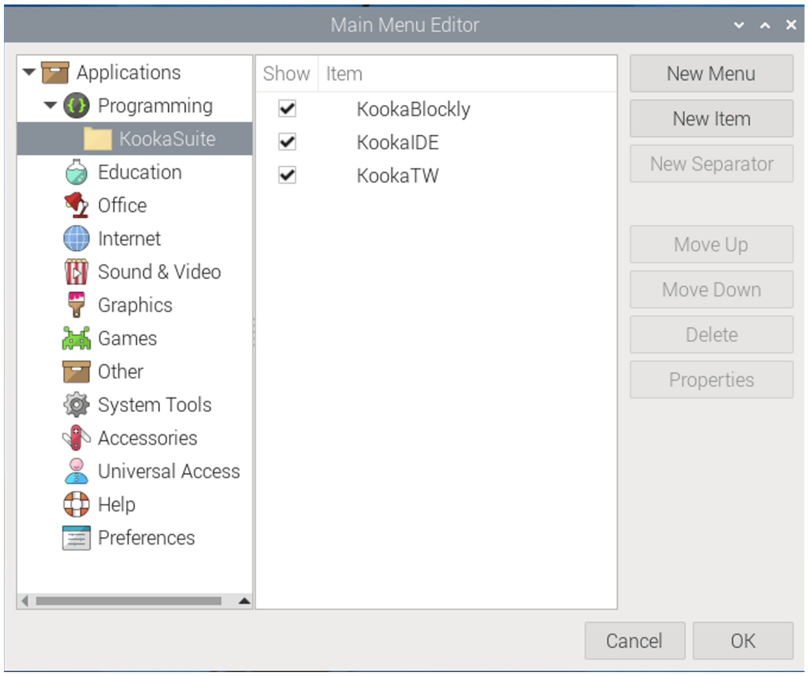


Fig. 1.13: Configuring **KookaSuite** apps using the **Raspberry Pi**'s menu editor.

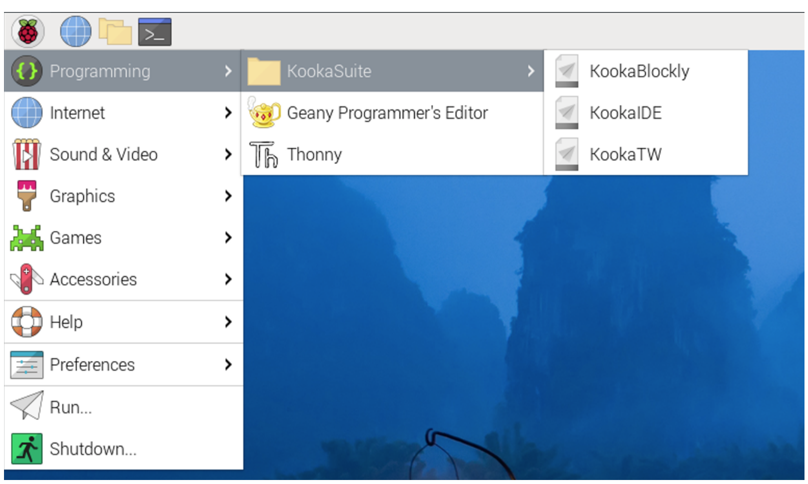


Fig. 1.14: The **KookaSuite** apps as they appear in the **Raspberry Pi**'s menu.

1.2.5 Script Folders

During installation or first running of **KookaSuite**, the `Kookaberry Scripts\` folder will be created in the location specified during the installation process or on MacOS and Raspbian in the user's home folder or documents folder.

If the `Kookaberry Scripts\` folder already existed it will not be altered. See Fig. 1.15.

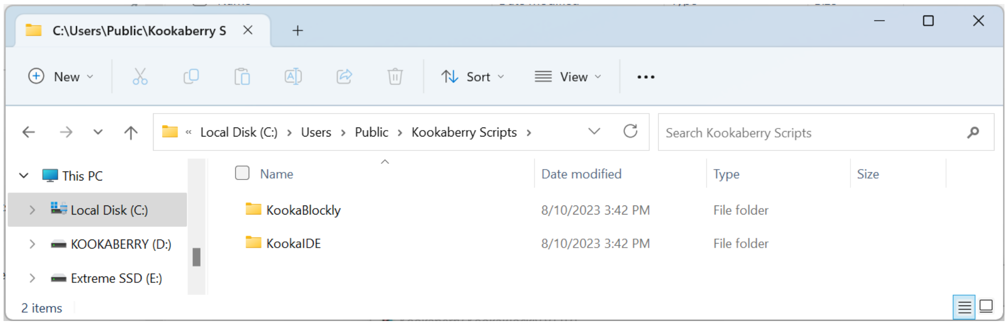


Fig. 1.15: The Kookaberry Scripts folder in a fresh **KookaSuite** installation.

The `Kookaberry Scripts\` folder contains two sub-folders:

- `KookaBlockly\` where **KookaBlockly** stores the program scripts created by it.
- `KookaIDE\` where **KookaIDE** stores MicroPython scripts.

It is permissible to create sub-folders within the `KookaBlockly\` and `KookaIDE\` folders for different projects.

The script selection drop-down boxes in **KookaBlockly** and **KookaIDE** will however only scan the first level of sub-folders for scripts.

1.2.6 KookaBlockly Updates

Occasionally when **KookaBlockly** updates are released, the forms and functions of some blocks may be changed.

Existing **KookaBlockly** scripts will retain the forms and functions of blocks as last edited. Updates to the blocks are not automatically applied to pre-existing scripts.

If the newer block is desired, then the **KookaBlockly** script must be edited and the block explicitly replaced by the newer form from the block palette.

Once an older block is removed it can no longer be used as it will no longer be available from the palette of blocks.

1.2.7 Editing KookaBlockly Scripts Using KookaIDE

A **KookaBlockly** file, designated with the file type suffix `.kby.py`, contains the MicroPython script that is automatically generated by the **KookaBlockly** editor as visual blocks are assembled and configured. At the end of the **KookaBlockly** file there is a very long comment line which contains the code, in XML (Extended Markup Language) format, that describes all the blocks, their parameters and their inter-connections.

While it is possible to edit a **KookaBlockly** file using the **KookaIDE** editor and to then run it on the Kookaberry, any changes made will not alter the XML block code. As soon as the **KookaBlockly** file is again opened by the **KookaBlockly** editor, it will regenerate the MicroPython script based on the XML block code, and it will disregard any changes made to the MicroPython script.

Attempting to edit the XML code directly will likely render the **KookaBlockly** file unusable by the **KookaBlockly** editor, so please do not edit the XML code.

Important: Only edit **KookaBlockly** files using the **KookaBlockly** editor!

1.3 Using the KookaBlockly Application

Launching **KookaBlockly** on a personal computer will result in the display shown in Fig. 1.16.

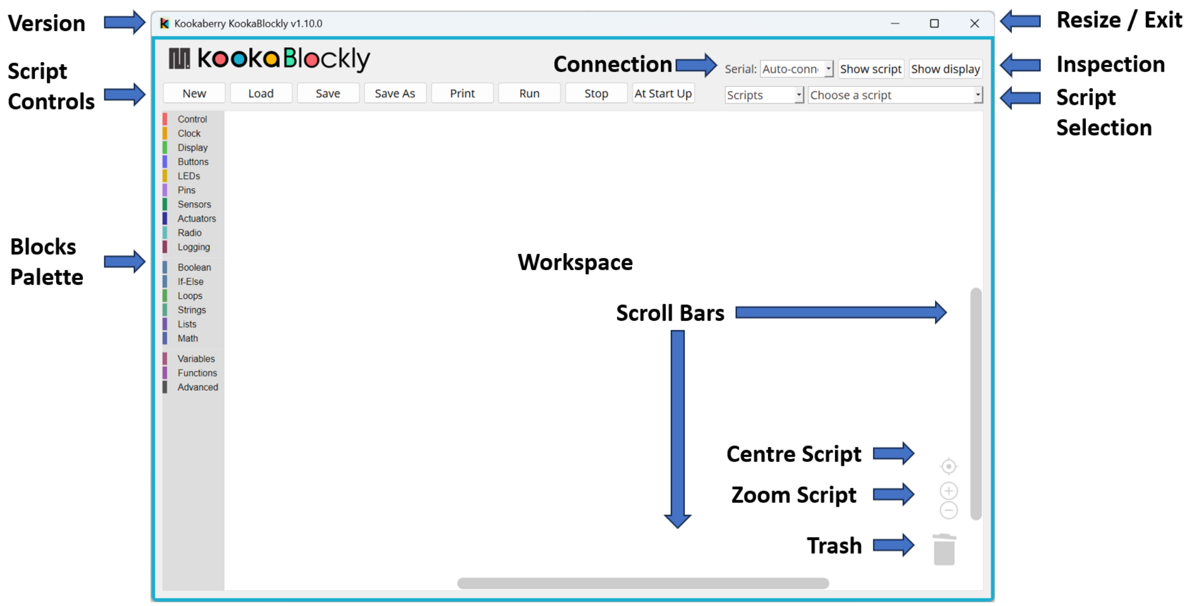


Fig. 1.16: This is the **KookaBlockly** display with the controls labelled.

The application window has numerous controls, as are described below:

1.3.1 Version

The version of **KookaBlockly** is shown at the top-left of the **KookaBlockly** window.

Note: The latest version of **KookaBlockly** can be conveniently downloaded from the **Kookaberry GitHub** repository at https://github.com/kookaberry/kooka-releases/releases.

See the section *Installing KookaBlockly* for instructions.

If a **KookaBlockly** script has been loaded, the path and name of the file from which the script was loaded is shown next to the **KookaBlockly** version.

1.3.2 Resize / Exit

These controls allow the **KookaBlockly** window to be minimises or maximised, and the KookBlockly application to be exited.

If the KookBlockly script has not been saved before attempting to exit **KookaBlockly**, a prompt dialogue will appear providing an opportunity to save or not save the current script to a file, as shown in Fig. 1.17.

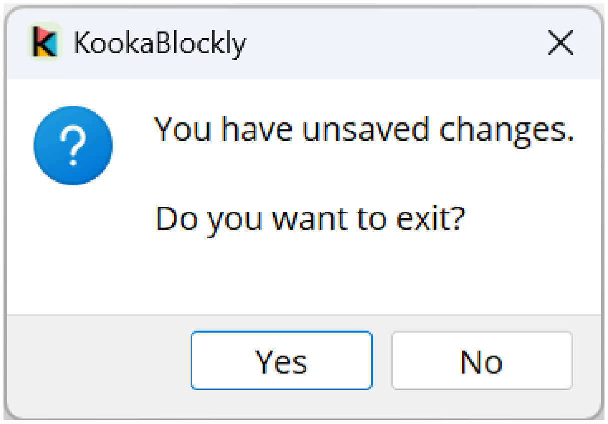


Fig. 1.17: Prompt dialogue on attempted exit with unsaved script.

Resizing of the window can also be accomplished by clicking on the window edges and dragging to resize.

The appearance and location of these controls varies between Windows, MacOS and Raspbian and conforms to the conventions used by the user interface of those operating systems.

1.3.3 Workspace

In the centre of the window is the **KookaBlockly** workspace.

Blocks can be dragged into this space, repositioned, resized and deleted by using the mouse or track-pad or pointing device.

1.3.4 Blocks Palette

Down the left of the window is a vertically-oriented list of the **KookaBlockly** palette categories, shown in Fig. 1.18.

Click on any category to reveal the palette of blocks, click on and drag the desired block to the workspace, position it and release to drop the block in place. The blocks palette will then automatically close.

To close the blocks palette without dragging a block into the workspace, either click on the palette icon used to open the palette, or press the `Esc` key.

The block categories and blocks are fully described in the *Part 2 - KookaBlockly Function Blocks Reference* section.

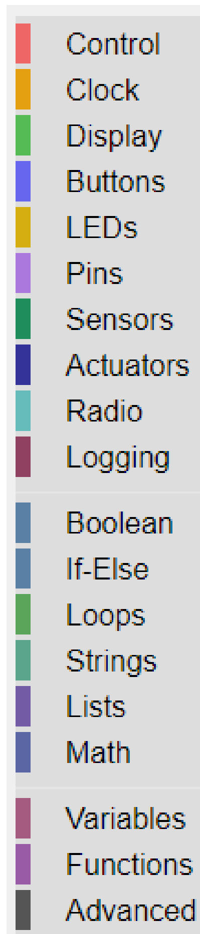


Fig. 1.18: The Blocks Palette showing the Block Categories

1.3.5 Script Controls

At the top-left of the window, a set of buttons with which **KookaBlockly** scripts may be created, loaded, saved, run and stopped. See Fig. 1.19.

Fig. 1.19: The **KookaBlockly** Script Control Buttons

The functions of each of the **KookaBlockly** Script Control buttons is:

New

Empties the workspace to start a new script. If the current script contents have not been saved then a save prompt is given as shown in Fig. 1.20.

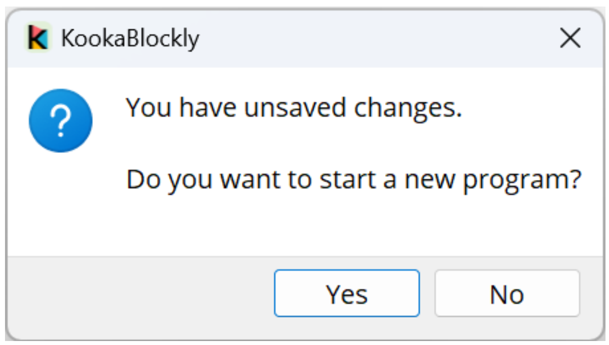


Fig. 1.20: Prompt dialogue on attempting to clear the workspace containing an unsaved script.

Load

The **Load** button allows the user to select a **KookaBlockly** program to be loaded into the Workspace, appending it to the current script. This feature enables the assembling of scripts by combining separate script files.

Move the cursor to this button, press click on the mouse and the dialogue in Fig. 1.21 will be displayed:

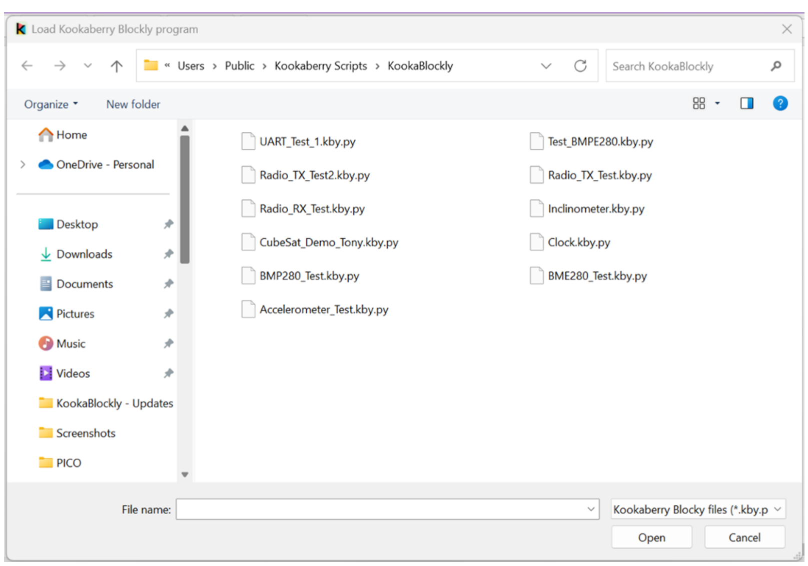


Fig. 1.21: **KookaBlockly** script load file selection dialogue.

The default directory for **Kookaberry** scripts within the current user's account is `/Kookaberry Scripts/KookaBlockly` and the user can navigate away from this as desired.

KookaBlockly script files have a type designation of `.kby.py`.

Selecting a script and pressing the dialogue's **Open** button, or alternatively double-clicking on a selected **KookaBlockly** script file will place a copy of that script in the **KookaBlockly** Workspace from where it can be modified, saved and run on the **Kookaberry**.

Important: When assembling scripts from a number of files, the name of the last loaded file becomes the default for saving the script. If the script is intended to be saved into a new or differently-named file then use the **Save As** button to give a different name to the file.

Save

Saves the currently named script to the corresponding file.

If the script was loaded from a file, the path and name of the file from which the script was loaded is shown next to the **KookaBlockly** version and the script will be save to that file.

If the script has not been previously saved, the **Save As** procedure is automatically used.

Save As

Saves the current script to a new file within a selected folder.

Move the cursor to this button, press click on the mouse and the file dialogue in Fig. 1.22 will be displayed:

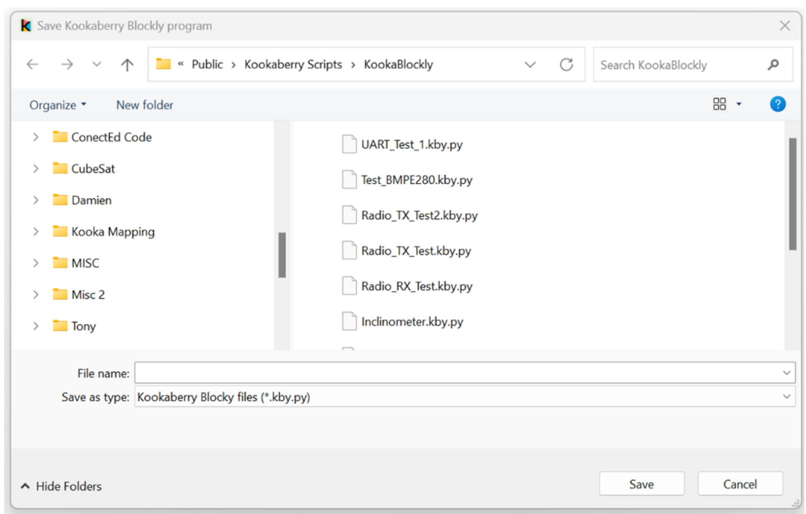


Fig. 1.22: **KookaBlockly** script save file selection dialogue.

The default directory for **Kookaberry** scripts within the current user's account is `/Kookaberry Scripts/KookaBlockly` and the user can navigate away from this to another folder as desired.

KookaBlockly script files have a type designation of `.kby.py`.

Enter the new file's name and press the dialogue's **Save** button will save the current script to the file. If the file already exists, another dialogue shown in Fig. 1.23 will open asking to confirm whether the file is to be replaced. Press **Yes** to overwrite the file, or **No** to go back and change the intended file name.

Print

Prints the current view of the script in the workspace, *which may not be the whole script*. Using the **Zoom** buttons and **Scroll Bars**, adjust the view of the script to suit the printout desired.

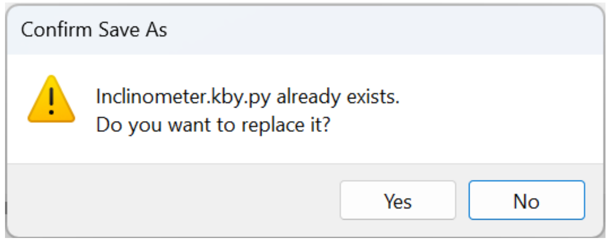


Fig. 1.23: **KookaBlockly** existing file name dialogue.

When the **Print** button is clicked, a Print dialogue (per the operating system convention) appears as in Fig. 1.24.

Choose the print options, which again are specific to the PC operating system and the installed printer, and then press the **Print** button to finalise printing options and then printing to the chosen printer.

Print options may include paper size, paper orientation, multi-page layout, printer selection and printer setup.

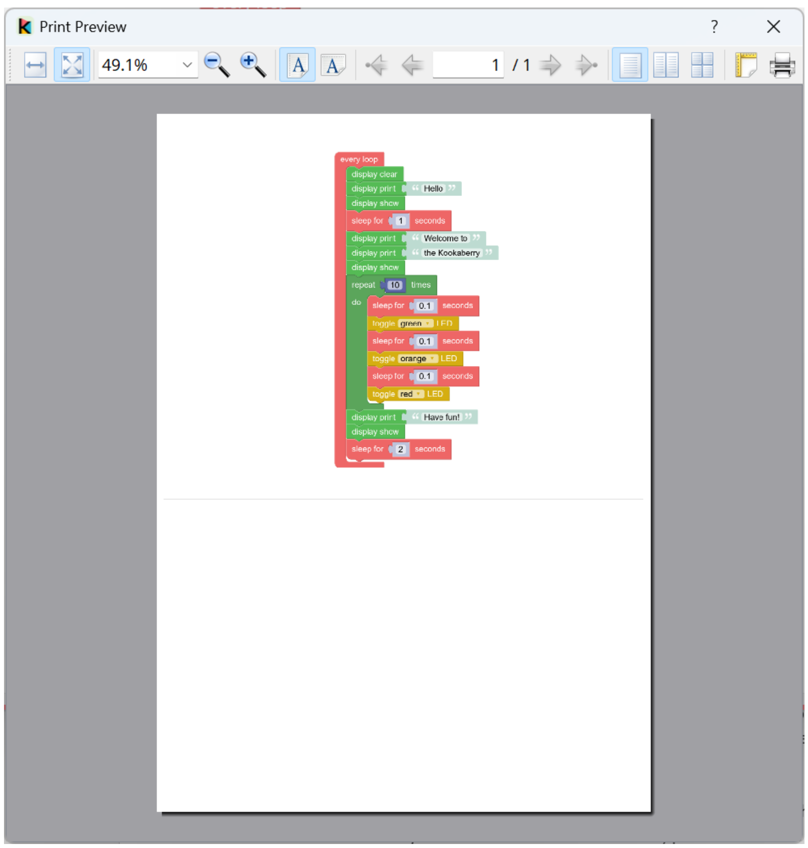


Fig. 1.24: **KookaBlockly** script Print dialogue.

Run
: Transfers the current script to the tethered **Kookaberry** and runs the script on the **Kookaberry**.

Stop
: Terminates the script currently running on the tethered **Kookaberry**.

At Start Up
: Gives the option to automatically run a script automatically whenever the **Kookaberry** is turned on or reset.

The **Kookaberry** will look for a script file called `main.py` in the root folder of its file store whenever it starts up. If the script is present, it will be run. Using the **At Start Up** button, a file called `main.py` is created containing a small script that causes a designated script in the **Kookaberry's** `app` folder to be run.

For this to work correctly, the script must first be stored on the **Kookaberry's** file storage system, in the `app` folder.

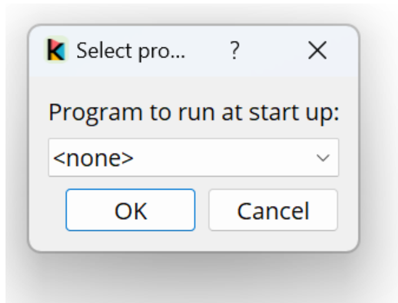


Fig. 1.25: **At Start Up** dialogue.

Click on the **At Start Up** button and a dialogue window, shown in Fig. 1.25, will appear with a drop-down list of the scripts stored on the **Kookaberry** as in Fig. 1.26.

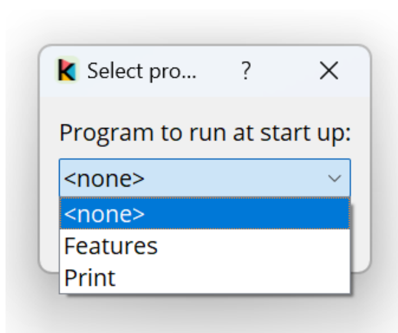


Fig. 1.26: **At Start Up** drop-down list of available scripts.

The first entry will be `<none>` followed by a list of scripts in the `app` folder.

Select the desired script and click the **OK** button.

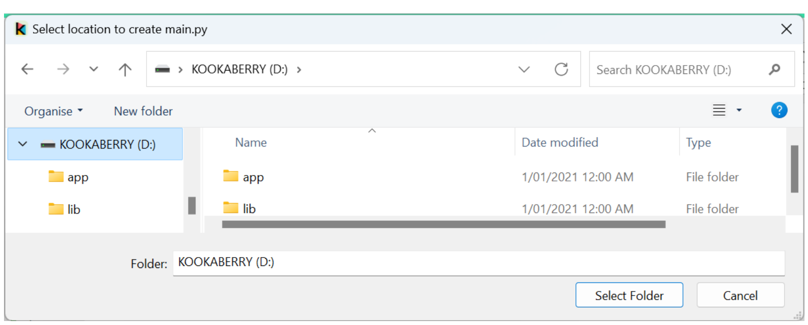


Fig. 1.27: **At Start Up** folder selection dialogue.

A folder dialogue window will then open, as in Fig. 1.27, to select where on the Kookaberry a script file called `main.py` should be stored. Usually this will be in the root folder of the **Kookaberry's** file store. However on occasion you may want to store the `main.py` file elsewhere. Select the folder and click on the **OK** button and the `main.py` file will be stored in the folder.

To stop the script from being automatically run, select `<none>` in the script selection dialogue and overwrite the previously stored `main.py`. A `main.py` file will still exist but without any instructions to start a script.

1.3.6 Inspection Buttons

At the top-right of the window, the Inspection Buttons will open separate windows.

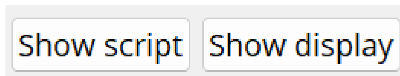


Fig. 1.28: The Inspection Buttons: Show script and Show display

Show display

This button which will open a window, shown in Fig. 1.29, on which the attached **Kookaberry** is shown in virtual form. This includes the **Kookaberry**'s display, **LEDs**, buttons A to D and reset, and a button to start the **Kookaberry**'s internal menu.

The display will mirror the physical display on the **Kookaberry**.

The **LEDs** will change colour to mirror illumination of the real **LEDs** on the **Kookaberry**.

The buttons can be clicked using a mouse or track-pad on the PC, and will respond in the same way as the physical buttons on the **Kookaberry**.

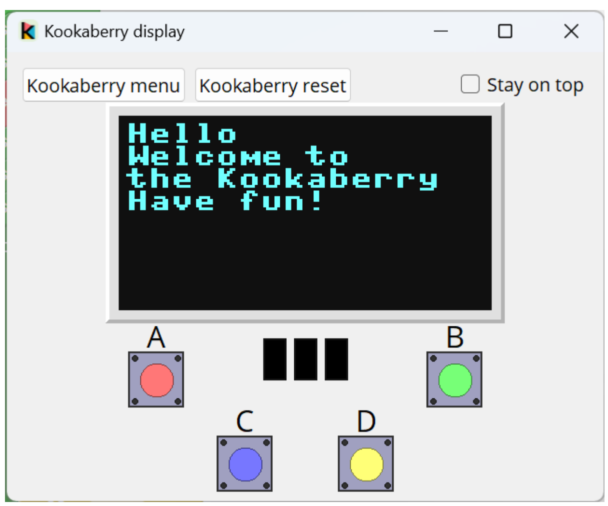


Fig. 1.29: Virtual **Kookaberry** window

Note: It is also possible to load **Kookaberry** firmware onto standard Pi Pico microcomputer boards. These boards do not have the physical **Kookaberry** display, LEDs or buttons.

In this case the virtual **Kookaberry** window can be used to view and operate the **Kookaberry**'s user interface.

1. the "Kookaberry Reset" button replicates the hardware Reset button the Kookaberry
2. the "Kookaberry menu" button replaces the "hold down button B and press and release Reset" on a physical Kookaberry
3. the three **LEDs** replicate the three hardware **LEDs** on the Kookaberry

4. the four buttons A, B, C and D, replicate the physical buttons on the KookaBerry

Show script

This button opens a window, shown in Fig. 1.30, in which the MicroPython script generated by the loaded **KookaBlockly** script is displayed.

The size of the window showing the script can be adjusted by clicking on and dragging the edges of the script window using the cursor.

The MicroPython is read-only and cannot be edited within this window.

There is a check-box which when ticked will cause the script window to stay visible in front of other windows on the computer screen.

This window presents a live view of the generated MicroPython script and it is possible to watch the MicroPython script being dynamically altered as the **KookaBlockly** script is being edited.

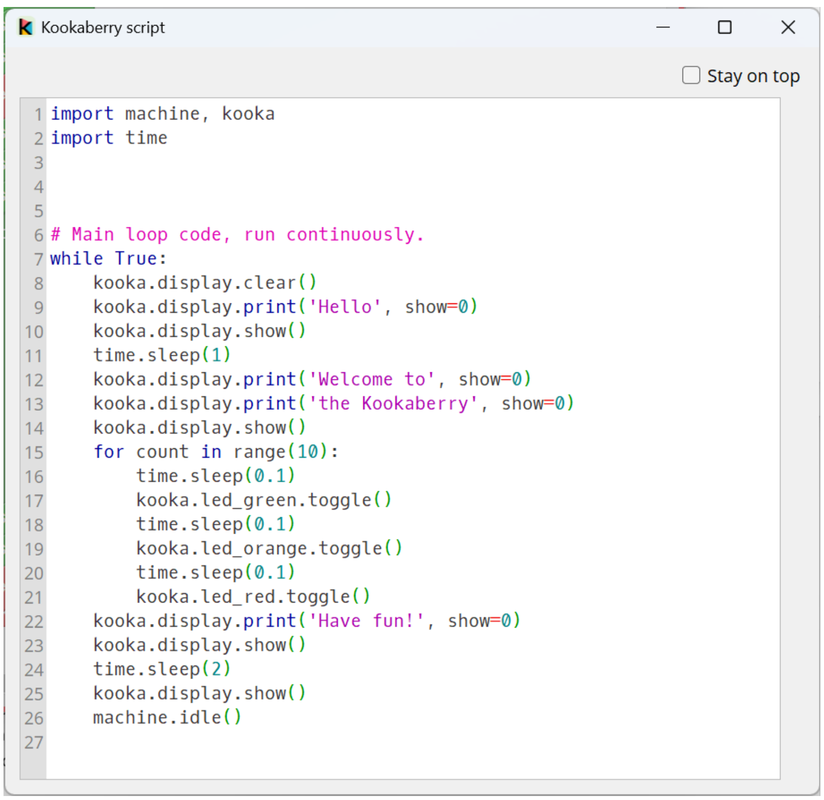


Fig. 1.30: **KookaBlockly**-generated MicroPython script window

1.3.7 Connection

At the top-centre is the "Serial" drop-down box which shows which serial USB ports are available and which is connected to a tethered **Kookaberry**. See Fig. 1.31.

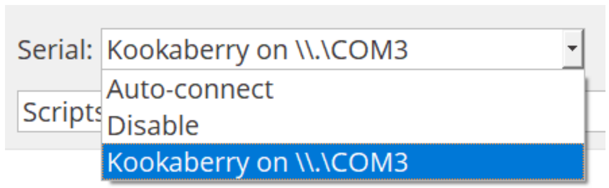


Fig. 1.31: The Serial drop-down showing the available and used USB serial connection ports

Plugging in a **Kookaberry** usually automatically assigns a USB serial port.

If the **Kookaberry** is not responding, select the `Auto-connect` option to reset the serial connection to the **Kookaberry**.

It is also possible to block a **Kookaberry** connection by selecting `Disable` from the dropdown-box.

1.3.8 Script Selection

Fig. 1.32: The Script Selection dropdown boxes

Scripts dropdown box
Shown in Fig. 1.32, this drop-down box contains a list of folders in the `Kookaberry Scripts/KookaBlockly` folder.

Choose a script
This contains a list of **KookaBlockly** scripts within the folder selected in the left-hand box.

Together these dropdown-boxes allow the selection and loading of any pre-existing KookBlockly script in the **KookaBlockly** folder and sub-folders.

If an unsaved **KookaBlockly** script is in the workspace, a prompt as shown in Fig. 1.33 will appear giving the opportunity to save the existing script to a file before replacing it with the selected script.

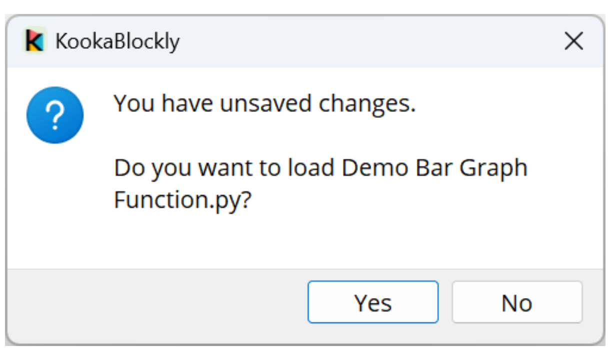


Fig. 1.33: Prompt dialogue on script replacement when an unsaved script is in the workspace.

1.3.9 Scroll Bars, Centre, Zoom and Trash

At the bottom-right of the window is a set of control icons as shown in Fig. 1.34.

Fig. 1.34: Control icons at the bottom right of the **KookaBlockly** window

Centre Script
: for centering the **KookaBlockly** script. Clicking on the Centre icon will centre the script in the Workspace and zoom it to fit the **KookaBlockly** window.

Zoom Script
: for changing the visual size of the **KookaBlockly** script by zooming in and out.

  Click on the + icon to zoom in and visually enlarge the script.

  Click on the - icon to zoom out and visually shrink the script.

Trash
: for retrieving blocks that were deleted during the current editing session.

  Click on the Trash icon to open it and show the blocks that have been deleted in the current editing session.

  To retrieve a block from the Trash, click on the block and drag it back into the Workspace.

  To close the Trash press the `Esc` key.

  When **KookaBlockly** is closed the contents of the Trash are deleted.

Scrollbars
: there are horizontal and vertical scrollbars for positioning the **KookaBlockly** workspace within the window.

  Click on a scrollbar and drag it up/down or left/right as appropriate to reposition the Workspace in the **KookaBlockly** window.

1.4 KookaBlockly Conventions

KookaBlockly provides an extensive palette of blocks to assemble into scripts. The blocks palette is on the left of the display organised into functionally related categories.

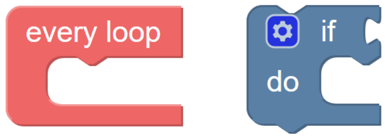
Clicking on a category, for example the **Control** category, reveals the blocks available within that category. To use the block, click on it and drag it onto the **KookaBlockly** workspace and release, and/or drag it into position until it snaps onto an adjacent block. Any block in the workspace can be clicked on and dragged into position.

The blocks palette will close automatically when a block is dragged into the workspace. Otherwise, the palette can be closed by clicking on the same block palette symbol that was used to open the palette, or by pressing the `Esc` key on the keyboard.

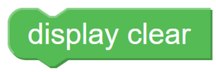
1.4.1 Block Shapes

KookaBlockly contains three basic block shapes:

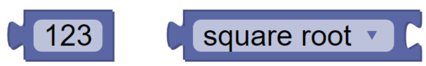
1. A C-shaped block directs program flow and contains a sequence of action blocks. The C-shaped block may be a loop, or may be a sequence of blocks that are run conditionally subject to one or more logical tests.



2. An action or "do" block which performs an operation. The block has an indent in the top border and a matching protrusion on the bottom border. These blocks click together like jigsaw pieces and may be placed in a vertical column and within a C-shaped block.

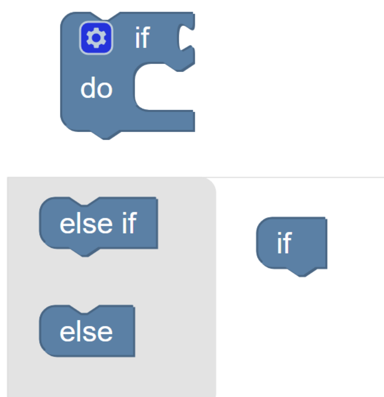


3. A value block which has a jigsaw tab on the left-hand edge. These blocks evaluate an expression and assign an output value to the blocks to which they are connected. Some value blocks have a matching receptacle on the right-hand edge which accepts other value blocks.



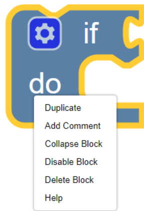
1.4.2 Block Configuration

Some blocks have configuration options denoted by a cog symbol. Clicking on the cog symbol presents options that may be used to configure the block.



1.4.3 Right-clicking

Right-clicking on a block also presents a set of option as below. These include: duplicate the current block; add a comment; collapse the block into a compact presentation or expand a collapsed block; disable or enable a block; remove the block from the program; or display some Help text about the block (if the Help text has been provided).



Duplicate

Click on Duplicate to create a duplicate of the block and any connected sub-blocks in the workspace.

Sub-blocks for example are all the blocks nested within a control block, or any value blocks connected to an action block.

Add Comment

Click on Add Comment and a circle with a question mark will appear in the block.

Click on the question mark and an area pane is provided for a user to enter in a comment.

This comment will be included in the MicroPython script generated by **KookaBlockly**.

Comments are very useful for describing parts or portions of the script for later reference by subsequent users of the script.

Collapse Block

Click on Collapse Block to truncate the block.

This is useful when a large number of blocks are in the workspace and the user wants to make a block smaller so that it is easier to see other blocks.

The user can restore the collapsed block at any time.

Disable Block

Click on Disable Block to make the block turn white and it will not be included in the script.

This is similar to "commenting out" lines of scripts when writing MicroPython code.

Delete Block

Choose a block by clicking on it.

Right click on the block and then choose Delete Block to delete the block from the script or press the Delete key on the keyboard.

Blocks can also be deleted by clicking on a block, separating it from the graphical script and dragging it into the Trash.

Clicking on the Trash icon, which is at the bottom-right of the Workspace, opens the lid and displays the deleted items.

Any deleted item may be dragged back into the workspace to become part of the program.

Clicking on a blank area of the workspace closes the Trash.

1.4.4 Text Delimiters

Many blocks contain text fields. In **KookaBlockly**, text is enclosed by double-quotes `"`, and these are automatically applied.

However there are some exceptions, particularly in the *Advanced* block which permits any valid MicroPython statement to be entered. Here it is important to use the double-quotes `"` and not single quotes `'` to delimit text, as single-quotes are used in **KookaBlockly**'s XML block code and will be misinterpreted rendering the saved **KookaBlockly** file unusable (without manually correcting the XML block code).

1.4.5 Deleting Blocks

Any block in the workspace, including any attached input blocks, can be removed from the script by:

1. dragging the block to the Trash at the bottom-right of the workspace. The Trash icon will show an open lid when the dragged block is correctly positioned.
2. or by clicking on the block to highlight it (shows a yellow outline), then pressing the `delete` key (or `backspace` key on Windows).

Blocks removed can be retrieved from the Trash by clicking on the Trash icon. A grey box will appear containing all of the deleted blocks. To retrieve a block, drag it back into the workspace. The Trash will then close automatically.

To close the Trash without dragging a block into the workspace, press on the `Esc` key.

CHAPTER

TWO

PART 2 - KOOKABLOCKLY FUNCTION BLOCKS REFERENCE

In this Part 2 of the **KookaBlockly Reference Guide**, each of the groups of function blocks available on the **KookaBlockly** palette are described in the following sections.

2.1 Control

The **Control** blocks in Fig. 2.1 direct program flow or provide time-related functionality.

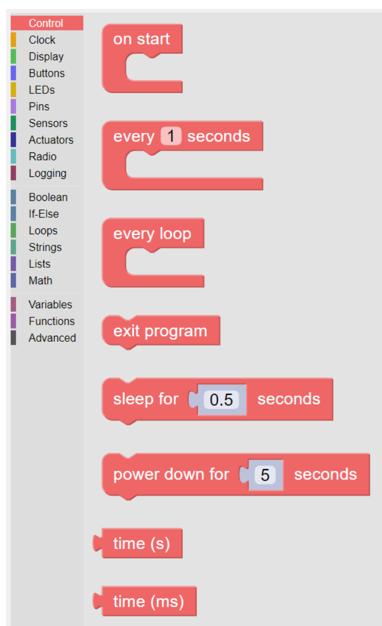


Fig. 2.1: The palette of **KookaBlockly Control** blocks

Each block is described in turn below.

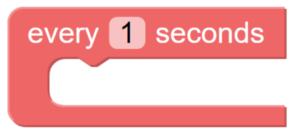
2.1.1 On Start

The "on start" block is intended to contain other action blocks that will run first and only once when the **KookaBlockly** script starts.

Typically the blocks contained are for the initialisation of the display, variables, sensors, and actuators.

2.1.2 Scheduled Loop

This block is a loop that repeatedly runs the blocks nested inside at the time interval specified in the numeric box.

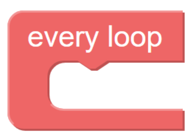


The loop will continue forever at the defined period unless the program is externally stopped.

The time specification is a number in decimal seconds, for example: 1 is 1 second, and 0.001 is 1 millisecond.

2.1.3 Every Loop

This block runs the blocks nested inside in a repeated loop.

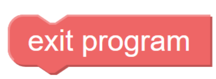


The loop will run forever unless externally stopped by exiting the script, or resetting the **Kookaberry** or removing power from the **Kookaberry**.

Another name for this block is the Repeat Forever loop.

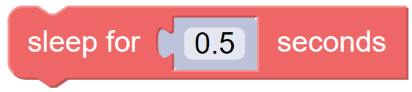
2.1.4 Exit Program

This block directs the running program to exit.



2.1.5 Sleep

This block causes the program to wait / pause for the specified time before continuing to the next block.



The number in the box specifies the duration of sleep in decimal seconds.

2.1.6 Time (s)

This block returns a value in whole seconds since the **Kookaberry's** epoch time (00:00:00 on 1st January 2000).

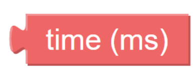
By subtracting successive values given by this block, the elapsed interval in seconds between the samples may be calculated which is useful for timing functions.

Note: **epoch time** is the point from which time is measured. This point differs for different operating systems. For MicroPython on micro-computers, the **epoch time** is 2000-01-01 00:00:00.

epoch time should not be confused with the **default time** set on the **Kookaberry**'s internal Real Time Clock (**RTC**), which is 2015-01-01 00:00:00. Using **KookaBlockly**, however, the **Kookaberry**'s internal **RTC** will be synchronised with the time on the PC it is tethered to using its USB connection.

2.1.7 Time (ms)

This block returns a value in milliseconds since the **Kookaberry's** epoch time (00:00:00 on 1st January 2000).



By subtracting successive values given by this block, the elapsed interval in milliseconds between the samples may be calculated which is useful for high-resolution timing functions.

2.2 Clock

Clicking on the **Clock** category in the **KookaSuite** palette reveals the available blocks, as in Fig. 2.2. Click and drag any of the required blocks to the **KookaBlockly** workspace and connect with other blocks to build a script that can use and/or set the time.

The blocks in the **Clock** category provide the functions to read and set the electronic real-time-clocks (**RTCs**).

The **Kookaberry** has an internal **RTC** which defaults to a time of 00:00:00 on 1 January 2015 when the **Kookaberry** is turned on.

The **Kookaberry** does not retain the time without external power as it has no internal battery to keep the internal clock running.

When the **Kookaberry** is connected to **KookaBlockly**, its internal **RTC** is updated to the time on the hosting computer.

Fig. 2.2: The palette of **KookaBlockly Clock** blocks.

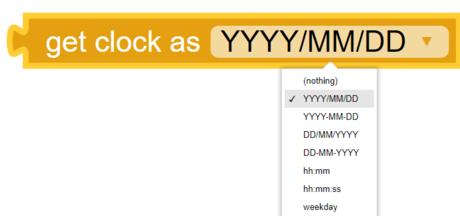
An external **RTC**, connected as an accessory to the **Kookaberry**, usually has a battery and therefore maintains the time that has been previously set on it. This provides a convenient way for the **Kookaberry** to obtain the correct time when it is not tethered to **KookaBlockly** (or **KookaIDE** or **KookaTW**). The external **RTC** is connected to the **Kookaberry** using two **Pins** specified as SCL and SDA on the relevant blocks.

Each of the **Clock** blocks is described in the following sections.

2.2.1 Internal Clock

Get Clock – Simple Time

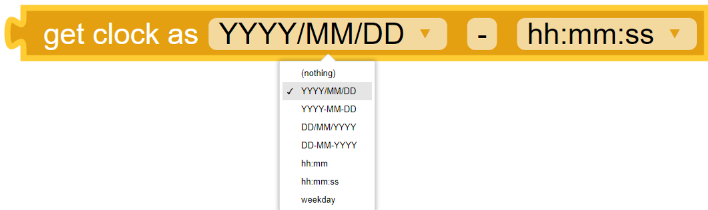
Reads the **Kookaberry's** internal Real Time Clock (**RTC**) and returns a date or time in the chosen format selected from the drop-down menu on the block.



The value returned is a character string.

Get Clock - Extended Time

Reads the **Kookaberry's** internal Real Time Clock (**RTC**) and returns the date and time as a character string comprising two parts per the selected formats and separated by a string of characters that can be specified by the user (the default separator is the minus character `-`).



In Fig. 2.3 is a **KookaBlockly** example script demonstrating a loop which updates the **Kookaberry's** display every second with the current time and date.

Fig. 2.3: A **KookaBlockly** Script that shows the current time and date on the **Kookaberry** display.

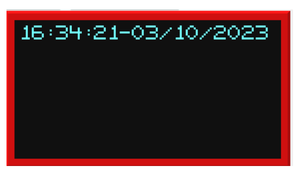


Fig. 2.4: The **Kookaberry** display resulting from the example **KookaBlockly** Script in Fig. 2.3.

Set Clock from Character String

This block sets the **Kookaberry's** internal Real Time Clock (**RTC**) to the time specified by a character string in the format "YYYY/MM/YY HH:MM:SS".

This is useful for updating the internal **RTC** with a fixed time or where the **Kookaberry** internal clock has not been automatically synchronised using **KookaBlockly**.

2.2.2 External Clock

External Clock's Pins Connections

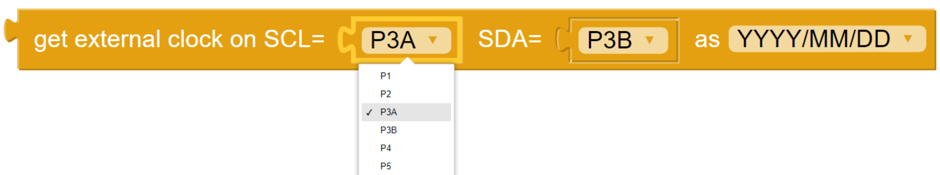
The external clock is connected to the **Kookaberry** by two of the five connectors on the back, `P1` through to `P5`, with connector `P3` having two possible connection points: `P3A` and `P3B`. (see the *Pins* category description).

The external clock block has two input **Pins** drop-down selection blocks by which the input Pin can be selected.

It is possible to replace the **Pins** dropdown selection block with a **String** block. This is useful when using **Pins** other than those exposed on the rear of the **Kookaberry**, or when other microprocessor boards that are compatible with **Kookaberry** firmware are being used. In those cases type in the Pin's identifier into the **String** block.

Get External Clock - Simple Time

Reads the **Kookaberry's** external Real Time Clock (**RTC**) and returns a date or time in the chosen format selected from the drop-down menu on the block.

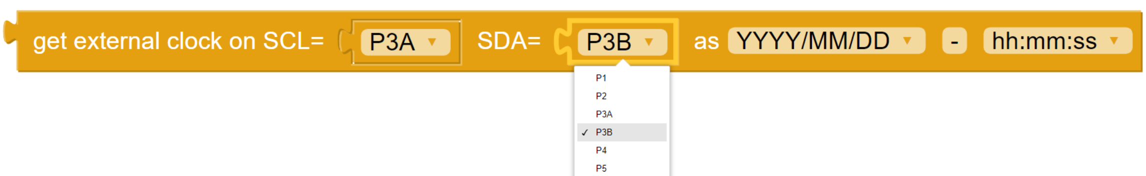


The value returned is a character string.

The external **RTC** is connected to the **Kookaberry**'s connector ports as selected from the SCL and SDA dropdown lists. The default setting of SCL as `P3A` and SDA as `P3B` is usually correct, meaning the external **RTC** is connected to the **Kookaberry** using the 4-pin P3 port.

Get External Clock – Extended Time

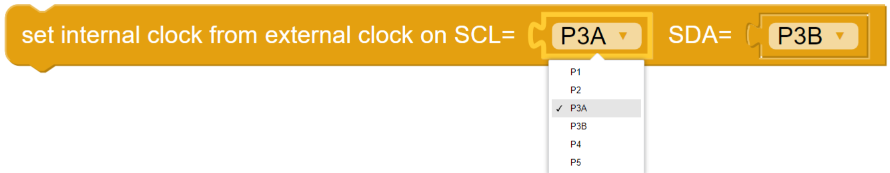
Reads the *Kookaberry's* external Real Time Clock (**RTC**) and returns the date and time as a character string comprising two parts per the selected formats and separated by a string of characters that can be specified by the user (the default separator is the minus character `-`).



The external **RTC** is connected to the **Kookaberry**'s connector ports as selected from the SCL and SDA dropdown lists. The default setting of SCL as `P3A` and SDA as `P3B` is usually correct, meaning the external **RTC** is connected to the **Kookaberry** using the 4-pin P3 port.

2.2.3 Set Internal Clock from External Clock

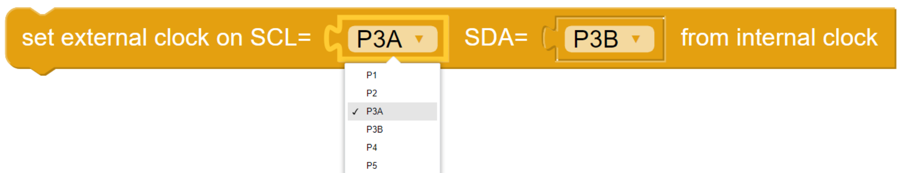
Sets the **Kookaberry's** internal Real Time Clock (**RTC**) by copying the current time from the external **RTC**.



The external **RTC** is connected to the **Kookaberry**'s connector ports as selected from the SCL and SDA dropdown lists. The default setting of SCL as `P3A` and SDA as `P3B` is usually correct, meaning the external **RTC** is connected to the **Kookaberry** using the 4-pin P3 port.

2.2.4 Set External Clock from Internal Clock

Sets the **Kookaberry's** external Real Time Clock (**RTC**) by copying the current time from the internal **RTC**.

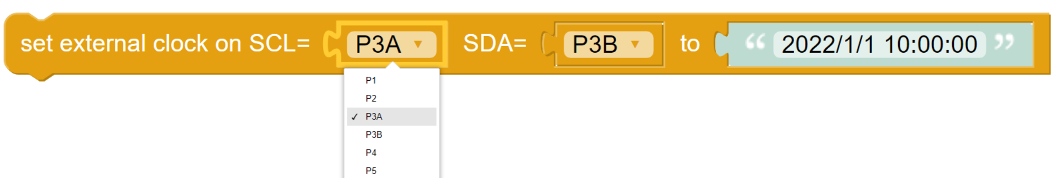


This is useful for updating the external **RTC** with the correct time when the **Kookaberry** is tethered to **KookaBlockly**.

The external **RTC** is connected to the **Kookaberry**'s connector ports as selected from the SCL and SDA dropdown lists. The default setting of SCL as `P3A` and SDA as `P3B` is usually correct, meaning the external **RTC** is connected to the **Kookaberry** using the 4-pin P3 port.

2.2.5 Set External Clock from Character String

Sets the **Kookaberry's** external Real Time Clock (**RTC**) to the time specified by a character string in the format "YYYY/MM/YY HH:MM:SS".



This is useful for updating the external **RTC** with a fixed time or where the **Kookaberry's** internal clock has not been automatically synchronised using **KookaBlockly**.

The external **RTC** is connected to the **Kookaberry**'s connector ports as selected from the SCL and SDA dropdown lists. The default setting of SCL as `P3A` and SDA as `P3B` is usually correct, meaning the external **RTC** is connected to the **Kookaberry** using the 4-pin P3 port.

2.3 Display

Display blocks in Fig. 2.5 control what appears on the **Kookaberry**'s display.

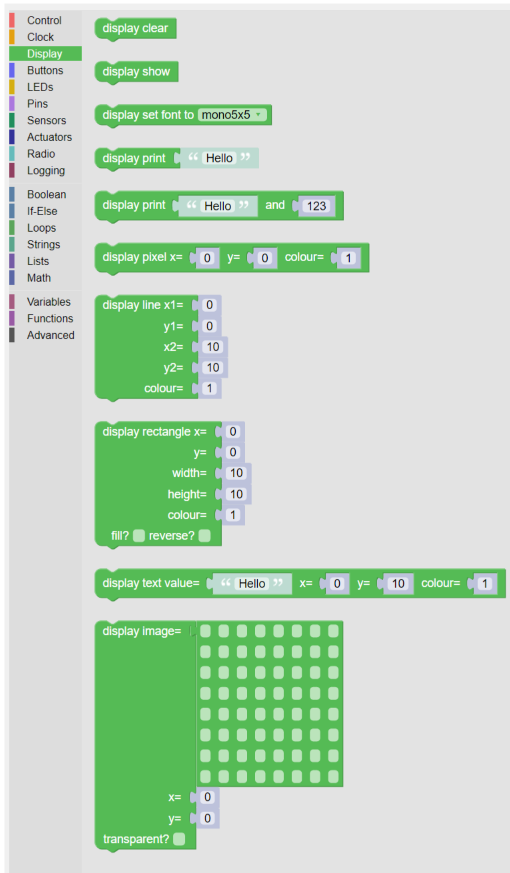


Fig. 2.5: The palette of **KookaBlockly Display** blocks

Each block is described in turn below.

2.3.1 Kookaberry Display

The **Kookaberry's** display is a 128 pixel wide x 64 pixel high cyan OLED (Organic Light Emitting Diode) display.

The x direction is the width of the display having a range specified as 0 to 127 pixels and the y direction is the height of the display having a range specified as 0 to 63 pixels.

As shown in Fig. 2.6, the (x,y) location (0,0) is at the top left-hand corner of the display. The bottom right of the display has a location reference (x,y) of (127,63).

The display is driven from an internal memory array known as a Framebuffer, into which the software writes the pixel data prior to its contents being transferred to the physical **Kookaberry** display. This reduces any display flicker.

The method of writing to a display is generally:

1. **Clear** the Framebuffer
2. Write text and/or graphics to the Framebuffer in one or more parts to build up the entirety of the **Display**'s contents, and then

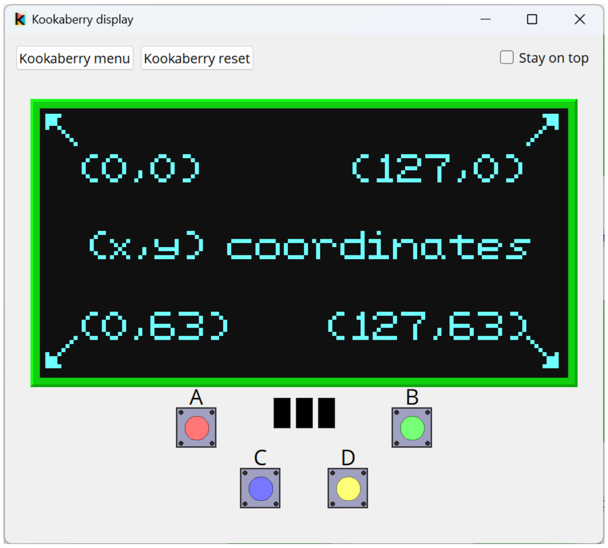


Fig. 2.6: The **Display** coordinates

3. **Show** the display buffer on the display.

The following blocks provide the functionality to operate the **Kookaberry**'s **Display**.

2.3.2 Text coordinates

The coordinates at which text is positioned on the **Display** differs from the graphical elements (`pixel`, `line`, `rectangle`, and `image`).

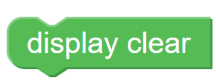
- Graphical elements are positioned at their top-left corner.
- Text is positioned at its bottom-left corner.

To accurately position text, one can use trial-and-error, or make a calculation that depends on the text font size (the default being `mono8x8`).

- To position a pixel at the top-left of the **Display** (0,0) simply specify x=``0`` and y=``0`` in the **Display Pixel** block.
- To position text at the top-left of the **Display**, specify (0,7) being x=``0`` and y=``7`` (the mono8x8 font height) in the **Display Print** block.

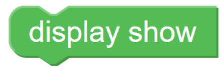
2.3.3 Display Clear

This block clears the display's frame buffer. The physical display will not be updated until a **Display Show** is used.



2.3.4 Display Show

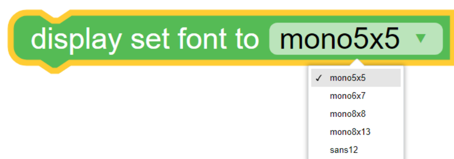
This block transfers the display's frame buffer to the **Kookaberry's** physical display.



KookaBlockly automatically inserts the equivalent **Display Show** code towards the end of the generated MicroPython script. However it may be desirable to refresh the physical display earlier in the **KookaBlockly** script, such as at the end of a loop that updates the display. Use this **Display Show** block in such circumstances as otherwise the display will not update until the end of the script.

2.3.5 Display Set Font

This block sets the character font to that selected from the drop down box.



The display fonts available for selection are from smallest to largest:

- `mono5x5` - each text character is 5 pixels wide by 5 pixels tall
- `mono6x7`,- 6 pixels wide by 7 pixels tall
- `mono6x8` - 6 pixels wide by 8 pixels tall
- `mono8x8` - 8 pixels wide by 8 pixels tall (the default font)
- `mono8x13` - 8 pixels wide by 13 pixels tall, and
- `sans12`.- 12 pixels wide by 12 pixels tall

The selected font will be applied from the point of selection.

A display using several fonts sizes may be constructed by using the **Display Set Font** block as the display Framebuffer is constructed by the **KookaBlockly** script.

2.3.6 Display Print

This block prints the editable text in the input value block to the **Kookaberry** display at position x=``0`` on a new line. The current line is set to the top of the screen immediately after the display is cleared.

If the line is longer than the display's width, the line is wrapped onto successive lines of the display. The current display line is increased by each successive **Display Print** until the bottom of the display is reached.

Thereafter each successive **Display Print** will scroll the display upwards by one line and the current line is shown at the bottom of the display.

2.3.7 Display Print-and

This block displays the editable text or value in the attached input value block on the current line of the display, followed by the output of any value block.

Fig. 2.7 shows an example to display the time:

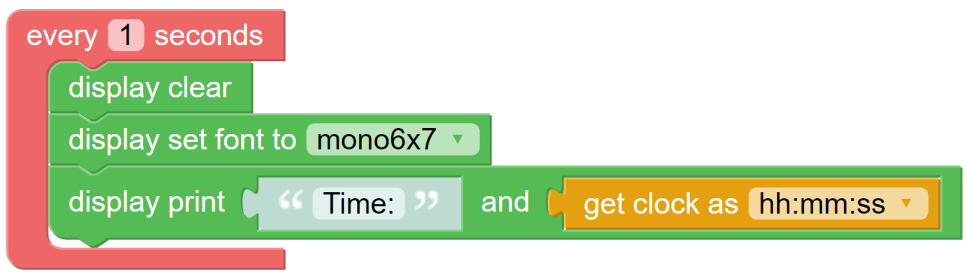


Fig. 2.7: **Display Print-and** example script

This example results in a display that looks like Fig. 2.8 and is updated every second.

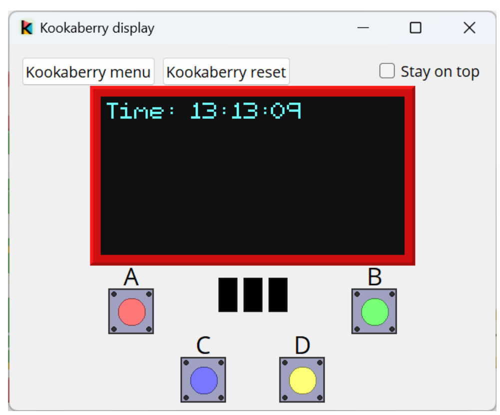


Fig. 2.8: **Display Print-and** example display

By using "Display Clear" the displayed text stays at the top of the screen instead of scrolling down the display.

2.3.8 Display Pixel

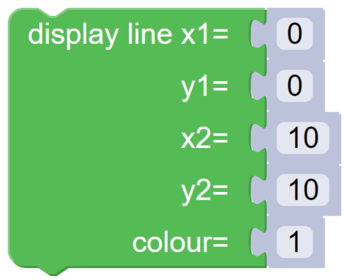
This block displays a pixel at the x and y locations with the specified colour on the display. The values of x, y and colour are the outputs of any value block.

If the values of x or y are outside of the display dimensions then the pixel will not be visible.

The values for colour should be either `0` or `1`, where `0` is pixel off (black) and `1` is pixel on (cyan).

2.3.9 Display Line

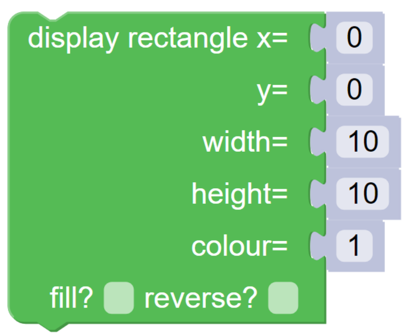
This block draws a line on the display starting from the location given by the values x1, y1 to the location given by the values x2,y2.



The value for colour should be either `0` or `1`, where `0` is pixel off (black) and `1` is pixel on (cyan).

2.3.10 Display Rectangle

This block displays a rectangle starting at location given by the values x, y with a width and height given by the results of the value blocks attached to those parameters.



The value for colour should be either `0` or `1`, where `0` is pixel off (black) and `1` is pixel on (cyan).

The **fill?** box when ticked fills the rectangle with pixels of the given colour.

The **reverse?** box specifies the orientation of the rectangle with respect to the x and y coordinates:

- if **reverse?** is not ticked, x and y specify the location of the top-left of the rectangle
- if **reverse?** is ticked, x and y specify the location of the bottom-right of the rectangle

The example script in Fig. 2.9 displays two rectangles of equal origin and dimensions, with one of them having the **reverse?** box ticked. The resulting display in Fig. 2.10 shows two rectangles, in normal and reverse orientations about the same x and y origin.

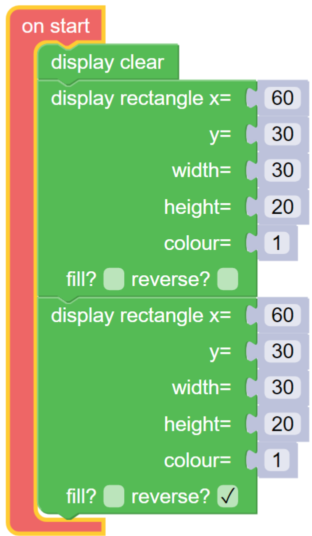


Fig. 2.9: Example showing the effect of the **reverse?** box on the **Display Rectangle** block

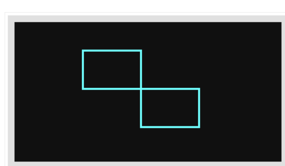


Fig. 2.10: The resulting display showing the effect of the **reverse?** box on the **Display Rectangle** block

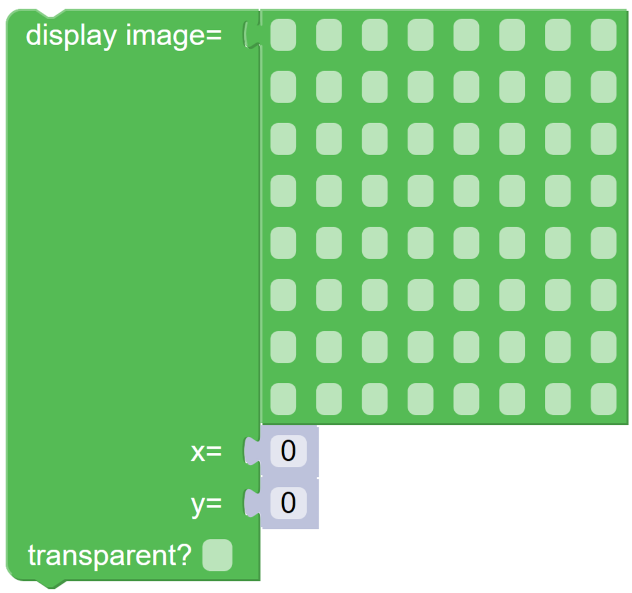
2.3.11 Display Text

This block enables the display of the attached output of the attached value block (ie "Hello") at the location specified by the value blocks at x and y on the display, with the colour being the value block output of 0 or 1.

Note: The (x, y) coordinate is where the bottom left corner of the display text is positioned.

2.3.12 Display Image

This block allows for the creation of an 8 x 8 pixel array positioned on the **Kookaberry** display at the coordinates of x and y.



The **transparent?** box if ticked will not extinguish any pixels that were already on, thereby giving an impression of transparency.

By manipulating the values of x and y using value blocks, the pixel array can be made to move around the screen.

Larger pixel arrays can be created by using multiple **Display Image** blocks with adjacent coordinates (by incrementing x and y in multiples of 8).

2.4 Buttons

Button blocks are used to specify the actions to be taken when a specific button is pressed. See Fig. 2.11.

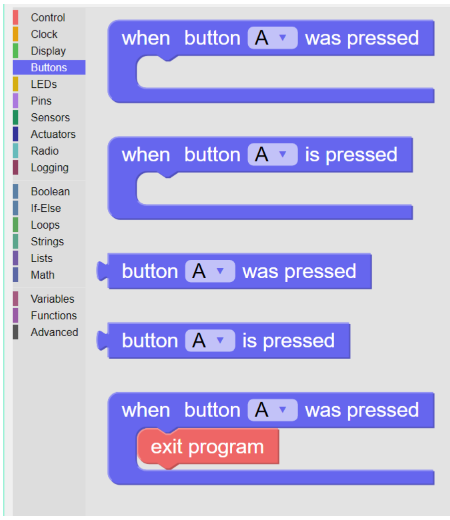


Fig. 2.11: The palette of **KookaBlockly Buttons** blocks

The **Kookaberry** has four buttons beneath the display labelled A, B, C and D.

These buttons are coloured `A` red, `B` green, `C` blue, and `D` yellow.

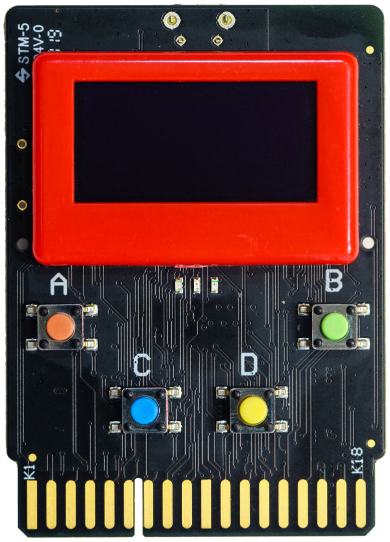


Fig. 2.12: **Kookaberry** - front view showing **Display**, **LEDs** and **Buttons**

Button functions are also available on the virtual **Kookaberry** which is shown when **KookaBlockly**'s **Show display** button is clicked.

Each block in the **Buttons** category is described in turn below.

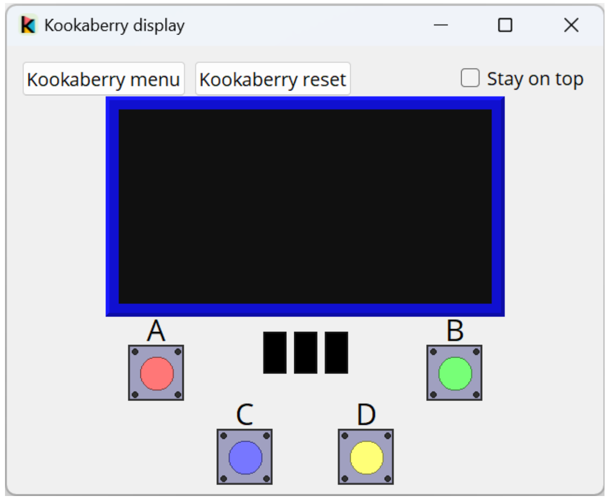
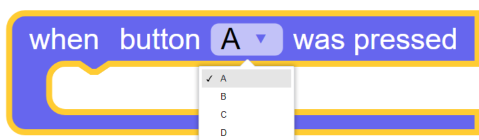


Fig. 2.13: Virtual **Kookaberry**

2.4.1 When Button Was Pressed

This is a control loop that performs the actions contained within it whenever the selected button *was pressed*.

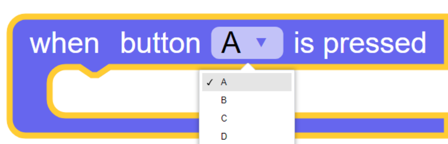


The button options are A, B, C, or D.

was pressed means that the actions within the loop will be performed only once after the selected button press.

2.4.2 When Button Is Pressed

This is a control loop that performs the actions contained within it as long as the selected button *is pressed*.

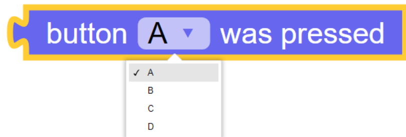


The button options are A, B, C, or D.

is pressed means that the actions will be performed repeatedly as long as the selected button is being pressed.

2.4.3 Button was pressed

This is a value block whose result is `True` (= 1) whenever the selected button was pressed.

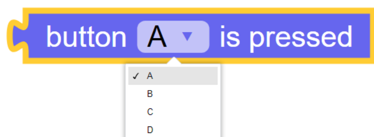


The button options are `A`, `B`, `C`, or `D`.

After this value block is used its output reverts to `False` (= `0`) until the next time the button was pressed.

2.4.4 Button is pressed

This is a value block whose result is `True` (= 1) as long as the selected button is being pressed.

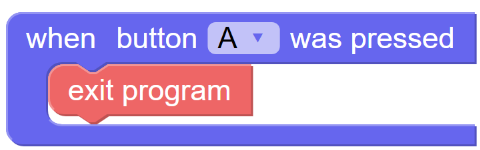


The button options are `A`, `B`, `C`, or `D`.

Th output of this value block reverts to `False` (= `0`) when the button is not being pressed.

2.4.5 Button to Exit Program

This is a combination of two blocks: the **button was pressed** control loop, as described above, and the **exit program** action.



The result of using this combination is whenever the button selected was pressed the currently running program will finish.

2.5 LEDs

The **LEDs** category, shown in Fig. 2.14, supports the three LED's that are beneath the display on the **Kookaberry**.

These **LEDs** are coloured `red`, `orange` and `green`.

In addition, support is provided for NeoPixel RGB **LEDs**.

Each block is described in turn below.

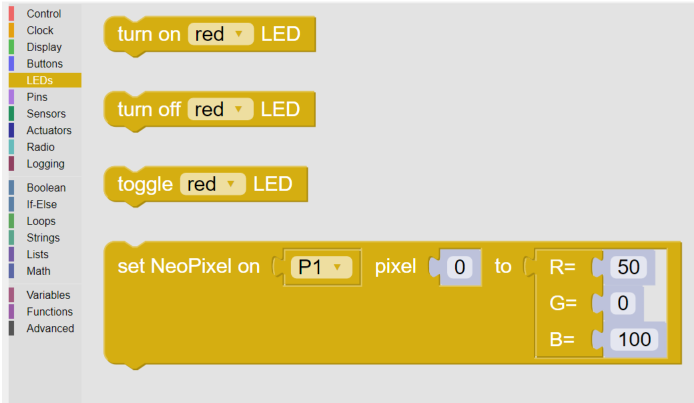


Fig. 2.14: The palette of **KookaBlockly** LED blocks

2.5.1 Turn ON LED

This block turns the LED, selected from the drop-down box, `ON`.

2.5.2 Turn OFF LED

This block turns the LED, selected from the drop-down box, `OFF`.

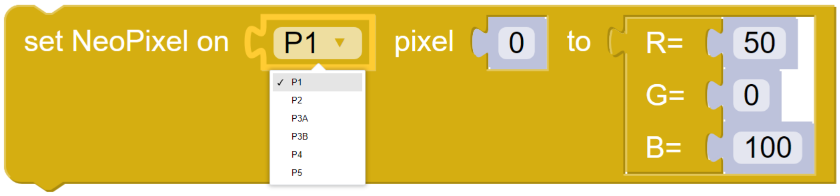
2.5.3 Toggle LED

This block toggles the LED selected in the drop-down box.

Toggle means to change the state of the LED from `OFF` to `ON`, or from `ON` to `OFF`, depending on the LED's state.

2.5.4 Set NeoPixel

This block supports NeoPixel arrays connected to one of the connections selected in the drop- down box.



Neopixels are multicolour **LEDs** with Red, Green and Blue **LEDs** in every individual Neopixel. The apparent colour of a Neopixel is the result of mixing the Red Green and Blue colours, in the same way that a television screen produces colours.

Neopixels come as single units or in chains of multiple Neopixels.

The following are the controls that can be set or manipulated on this block:

Pin

The **Kookaberry** has five connectors on the back, `P1` through to `P5`, with connector `P3` having two possible connection points: `P3A` and `P3B`. (see the *Pins* category description).

It is possible to replace the **Pins** dropdown selection block with a **String** block. This is useful when using **Pins** other than those exposed on the rear of the **Kookaberry**, or when other microprocessor boards that are compatible with **Kookaberry** firmware are being used. In those cases type in the Pin's identifier into the **String** block.

pixel

This is an integer commencing at `0` which specifies which pixel in the array will be set.

Important: The **Kookaberry** can only supply a limited amount of current power to a NeoPixel array. It is recommended to use no more than 8 NeoPixels, and also to limit the brightness of each to no more than `50` when using more than 4 NeoPixels.

If more NeoPixels and/or brighter illumination is required, then a special power adapter between the **Kookaberry** and the NeoPixel array is recommended.

RGB values

Each of the R (red), G (green) and B (blue) values can be set with integers in the range `0` to `100` inclusive.

By varying the ratio of RGB values set, a wide range of colours can be achieved, as shown in Fig. 2.15.

Learn more about using NeoPixels here: https://learn.auststem.com.au/peripheral/rgb-led/

2.6 Pins

The **Pins** category, shown in Fig. 2.16, provides the means to control what the **Pins** do.

Pins are electrical connectors on the **Kookaberry**.

The **Kookaberry** circuit board has five plugs on the rear numbered `P1` to `P5`.

`P3` has four electrical pins and the rest have 3 pins.

Fig. 2.15: RGB Primary Colour Combinations

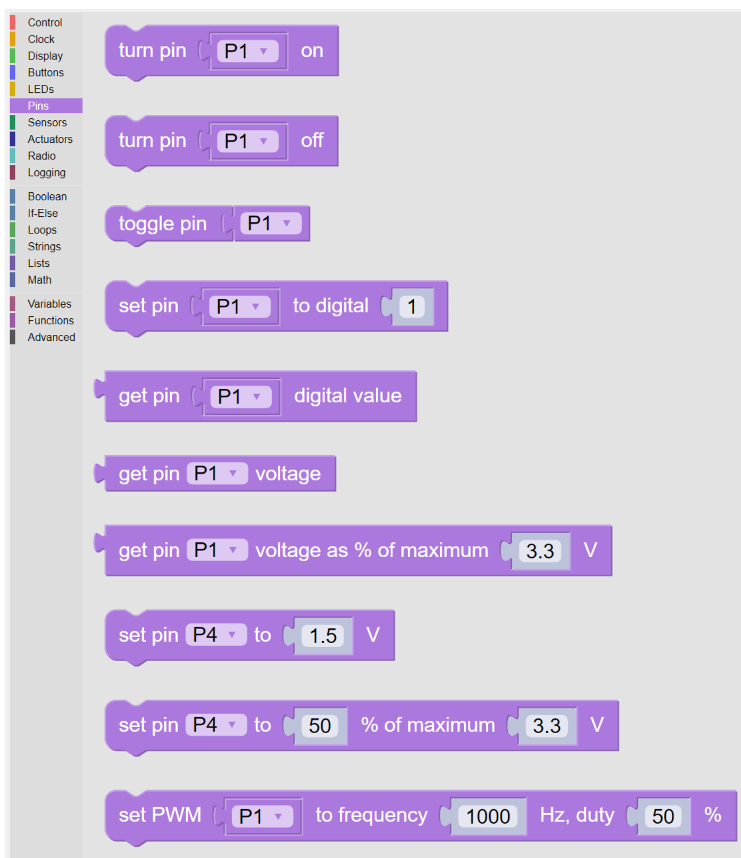


Fig. 2.16: The palette of **KookaBlockly Pins** blocks

On each connector two of the pins are used for positive and negative power connections. The remaining pin(s) have multiple uses as digital or analogue inputs or outputs.

In some of the **Pins** blocks it is possible to replace the drop-down selection block with a **String** block. This is useful when using **Pins** other than those exposed on the rear of the **Kookaberry**, or when other microprocessor boards that are compatible with **Kookaberry** firmware are being used. In those cases type in the Pin's identifier into the **String** block, as shown in Fig. 2.17.

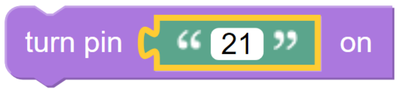
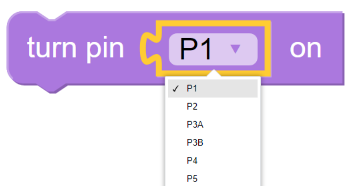


Fig. 2.17: Using a **String** value block instead of a **Pins** drop-down selection.

There are break-out (expander) circuit boards for the **Kookaberry** and the **Pi Pico** that make more of the **GPIO Pins** available for connection and therefore practical use within **KookaBlockly** scripts.

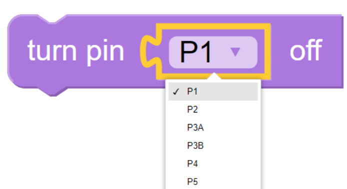
2.6.1 Pin Turn ON

The Pin Turn ON block causes the selected pin to behave as a digital output and to be turned on with an output voltage of +3.3 volts DC.



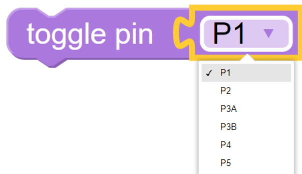
2.6.2 Pin Turn OFF

The Pin OFF block causes the selected pin to behave as a digital output and to be turned off with an output voltage of 0 volts DC.



2.6.3 Pin Toggle

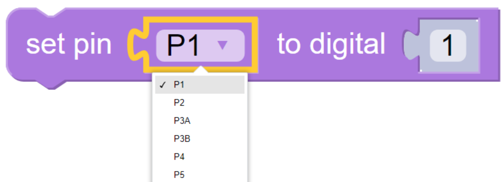
The Pin Toggle block causes the selected pin to behave as a digital output and to change state from `OFF` to `ON`, or from `ON` to `OFF`, depending on its existing state.



`OFF` sets the Pin to 0 volts DC, and `ON` sets the Pin to +3.3 volts DC.

2.6.4 Set Pin to Digital Value

The Pin Set Pin Digital Value block causes the selected pin to be set to according to the integer value of the input block.

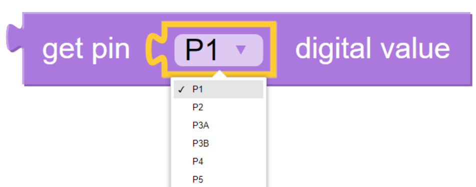


If the input value is `0`, the output of the Pin will be set to `OFF` which is 0 volts DC.

If the input value is not `0`, typically `1` or greater, then the output of the Pin will be set to `1` which is +3.3 volts DC.

2.6.5 Get Pin Digital Value

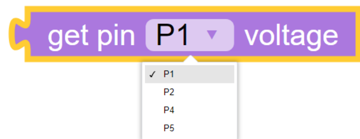
This value block designates the selected pin as a digital input and returns the digital value of the input as either `0` if the input voltage is close to 0 volts DC, or `1` if the input voltage is closer to +3.3 volts DC.



Important: The allowable **Pin** input voltage range for the **Kookaberry** is 0 volts to +3.3 volts DC. Applying voltages outside that range may irreparably damage the **Kookaberry**.

2.6.6 Get Pin Voltage

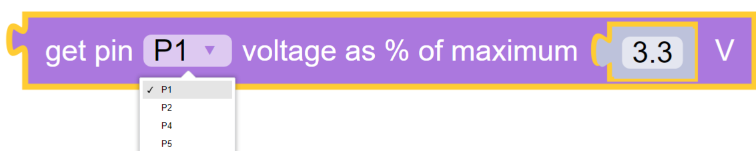
This value block designates the selected pin as an analogue input and returns a floating point value of the input in volts DC.



Important: The allowable **Pin** input voltage range for the **Kookaberry** is 0 volts to +3.3 volts DC. Applying voltages outside that range may irreparably damage the **Kookaberry**.

2.6.7 Get Pin Voltage as Percentage of Maximum

This value block designates the selected pin as an analogue input and returns an integer percentage value of the allowable **Kookaberry** input voltage range.



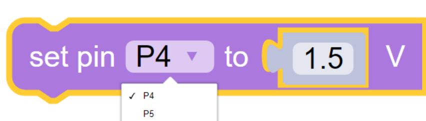
Applying 0 volts DC to the input Pin will resturn a value of `0`.

Applying +3.3 volts DC to the input Pin will resturn a value of `100`.

Important: The allowable **Pin** input voltage range for the **Kookaberry** is 0 volts to +3.3 volts DC. Applying voltages outside that range may irreparably damage the **Kookaberry**.

2.6.8 Set Pin to Voltage

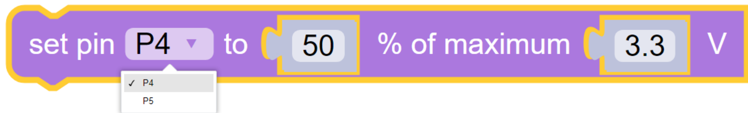
Where available on the **Kookaberry** the Set Pin to Voltage block causes the selected pin to behave as an analogue output and to be set to the voltage specified by the input block.



Note: **Set Pin to Voltage** is not available on **Kookaberry** using the Raspberry **Pi Pico RP2040** microprocessor.

2.6.9 Set Pin to Percentage of Maximum

Where available on the **Kookaberry** the Set Pin to Percentage of Maximum block causes the selected pin to behave as an analogue output and to be set to the percentage of maximum voltage specified by the input block.

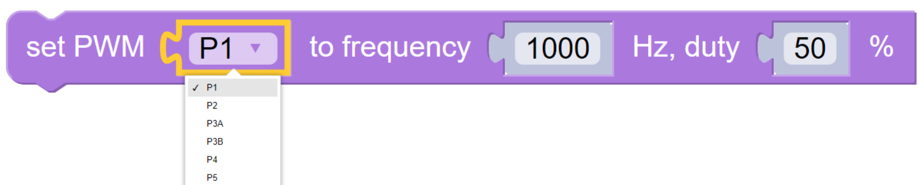


The output voltage will rise from 0 volts DC to +3.3 volts DC linearly with the input block rising from `0` to `100`.

Note: Set Pin to Percentage of Maximum is not available on **Kookaberry** using the Raspberry **Pi Pico RP2040** microprocessor.

2.6.10 Pin – Pulse Width Modulation (PWM)

Pulse Width Modulation (PWM) oscillates the selected Pin as a digital output between `0` (0 volts) and `1` (+3.3 volts DC) at a given frequency and duty cycle as specified in the input blocks.



The duty cycle is the proportion of each oscillation in which the output state is set to 1. A duty cycle of 50 means that the oscillation is `0` for 50% of the time and *1* for the remaining 50%.

The frequency is the number of times the output cycles per second. Frequency can be any positive floating point value

Both frequency and duty can be derived from other value blocks or specified directly.

PWM is used to apply speed control to DC motors by varying the duty cycle from 0% (motor is stopped) to 100% (motor at full speed). Additional circuitry is required to deliver the electrical power that a motor requires.

PWM can also be used to play tones through a loudspeaker by varying the frequency according to the pitch required. A frequency of 440Hz corresponds to the musical note of middle A on a piano, for example. Duty cycle is usually set to 50% but other interesting harmonics may be produced by varying the duty cycle over a limited range around 50%. Additional circuitry is also required to successfully drive a loudspeaker.

See also https://en.wikipedia.org/wiki/Pulse-width_modulation

Important: Please note that motors and loudspeakers should not be directly plugged into a **Kookaberry** connector. These devices require special electronics to supply more power.

Plugging in motors or loud speakers without the necessary driving electronics may irreparably damage the **Kookaberry**.

2.7 Sensors

The **Sensors** category provides blocks that enable the use of these sensors, as shown in Fig. 2.18.

Fig. 2.18: The palette of **KookaBlockly Sensor** blocks

The **Kookaberry** contains two on-board sensors, being a 3-axis accelerometer and a 3-axis magnetometer.

A large variety of external sensors may also be connected to the **Kookaberry** via its **Pin** connectors.

KookaBlockly supports many external sensors as are listed under the **External Sensors** section. These encompass measuring temperature, humidity, barometric pressure, soil moisture, light, electrical power, voltage and current.

2.7.1 Internal Sensors

Get Accelerometer (raw)

The **Kookaberry** contains an internal 3-axis accelerometer.

The accelerometer block provides the acceleration value of the selected axis (one of the X, Y and Z axes in the sensor's frame of reference), or the magnitude of the vector sum of all the axes. The X, Y and Z axes are selected using the drop-down list on the right of the block. The values are in metres per second squared.

The **Kookaberry's** internal accelerometer is oriented so that the X axis is along the horizontal dimension of the display, the Y axis is aligned with the vertical dimension of the display, and the Z axis is perpendicular to the **Kookaberry's** circuit board.

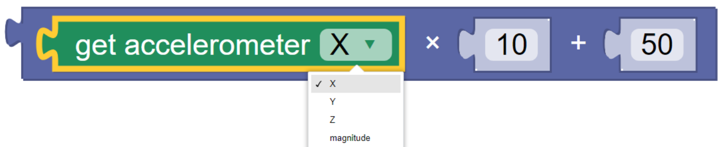
A typical value for acceleration is due to the earth's gravity, being 9.81 m/sec^2. This will vary slightly with geographic latitude and height as distances from the earth's centre of mass vary.

Note: The vector sum of all acceleration axes is the square root of the sum of the squares of the three axes. That is sqrt(x^2 + y^2 + z^2).

See also See https://www.explainthatstuff.com/accelerometers.html

Get Accelerometer (scaled)

The scaled accelerometer compound block is a convenient combination applying a multiplier and an offset to the raw accelerometer reading.

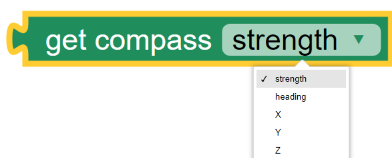


The scale and offset factors can be typed in directly or provided by plugging in other value blocks.

This block is useful to adjust the sensitivity of the accelerometer and to compensate for offsets such as the ever-present acceleration due to gravity.

Get Compass

The **Kookaberry** has an internal 3-axis magnetometer which can measure the magnetic field strength it is subjected to in three axes (X, Y and Z), as well as the total magnetic field strength, and the compass heading.



- The readings for magnetic field strength are in Gauss.
- The reading for `heading` are in degrees in the range `0` to `359` with `0` being North

See also https://en.wikipedia.org/wiki/Magnetometer

2.7.2 External Sensors

Sensors' Pins Connections

External sensors are connected to the **Kookaberry** by one of the five connectors on the back, `P1` through to `P5`, with connector `P3` having two possible connection points: `P3A` and `P3B`. (see the *Pins* category description).

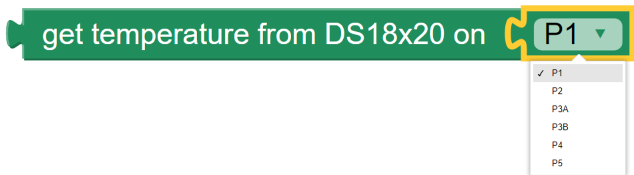
Each external sensor block has one or more input **Pins** drop-down selection blocks by which the input **Pin** can be selected.

It is possible to replace the **Pins** dropdown selection block with a **String** block. This is useful when using **Pins** other than those exposed on the rear of the **Kookaberry**, or when other microprocessor boards that are compatible with **Kookaberry** firmware are being used. In those cases type in the **Pin**'s identifier into the **String** block.

Get Temperature from DS18x20

The DS18x20 Probe is a waterproof digital temperature sensor that can measure temperature from -55°C to + 125°C with an accuracy of 0.5 ° C.

This block enables reading of the probe and returns the temperature in degrees centigrade. The drop-down box on this block enables selection of which **Pin** connector the sensor is attached to.



The DS18x20 sensor is used for measuring temperature in air and in liquid.

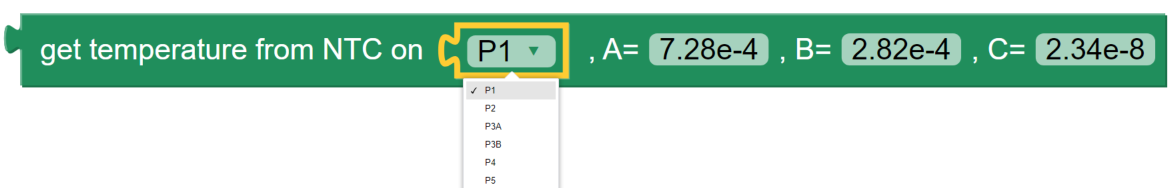
The sensor is pre-calibrated and performs all of the temperature calculations within the sensor.

Learn how to use the sensor here: https://learn.auststem.com.au/peripheral/ds18b20/

Note: The manufacturer of the temperature sensing DS18x20 chip requires a 4700 ohm (often referred to as a 4K7) pull-up resistor to make the chip work correctly. The **Kookaberry**'s and **Pi Pico**'s internal pull up resistor may work on some DS18x20 chips but not all of them. Try adding a pull-up resistor between the digital input **Pin** and **Vcc** by means of a pull-up adapter module, or use a different make of DS18x20 sensor if troublesome operation occurs.

Get Temperature from NTC

The NTC (Negative Temperature Coefficient) thermocouple sensor works through measuring its resistance which reduces as temperature rises. The **Kookaberry** performs the necessary calculations to convert the sensor's resistance to a temperature reading in degrees centigrade.



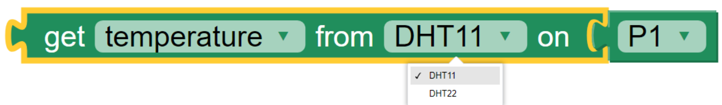
The options on the NTC value block are:

- The connector to which the sensor is attached
- The parameters A, B and C are the coefficients used in the Stein-Hart equation that is used to convert thermocouple resistance to temperature. Explaining this in more depth is beyond the scope of this manual. It is recommended that the default values not be altered.

See also https://www.explainthatstuff.com/howthermocoupleswork.html for an explanation of thermocouples.

Get Temperature or Humidity from DHT11 or DHT22

The **Kookaberry** supports the DHT11 and DHT22 temperature and humidity sensors. This block obtains the value of the selected parameter from the DHT sensor.



The drop-down boxes on the DHT value block permit the selection of:

- the sensor reading to be returned: temperature (in degrees Centigrade) or relative humidity (as a percentage)
- the sensor type being used: DHT11 or DHT22
- the connector to which the sensor is connected.

The DHT sensors are only suitable for measuring air temperature.

The difference between the two sensor types is that the slightly more expensive DHT22 sensor has a higher level of accuracy and precision.

- the DHT11 temperature range is from 0 to 50 degrees Celsius with +-2 degrees accuracy.
- the DHT11 humidity range is from 20 to 80% with 5% accuracy.
- the DHT22 temperature measuring range is from -40 to +125 degrees Celsius with +-0.5 degrees accuracy.
- the DHT22 humidity measuring range is from 0 to 100% with 2-5% accuracy.

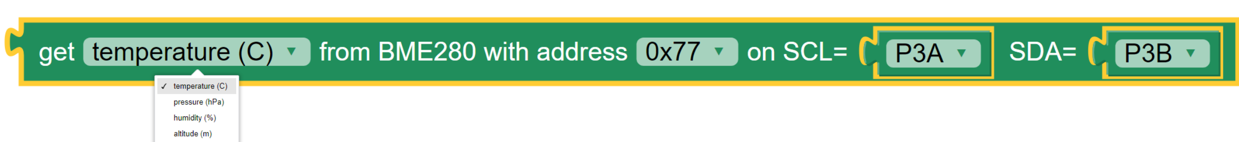
Please be sure to select the type of DHT sensor that matches the connected sensor or else erroneous readings will result.

The manufacturers of the DHT11 and DHT22 sensors recommend an interval between successive readings of no less than 2 seconds. Attempting shorter intervals will result in no reading and could also cause the **Kookaberry** script to terminate.

Learn more about using the DHT11 here: https://learn.auststem.com.au/peripheral/dht11/ and the DHT22 here: https://learn.auststem.com.au/peripheral/dht22/

Get Temperature / Humidity / Pressure from BME280

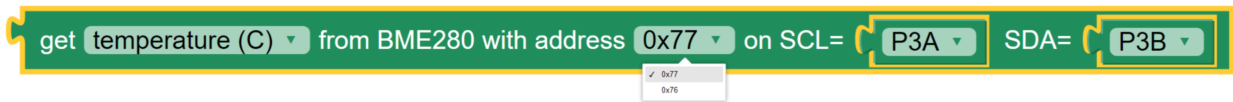
The **Get Temperature from BME280** block is shown below with the three sets of options available from the drop-down boxes on the block.



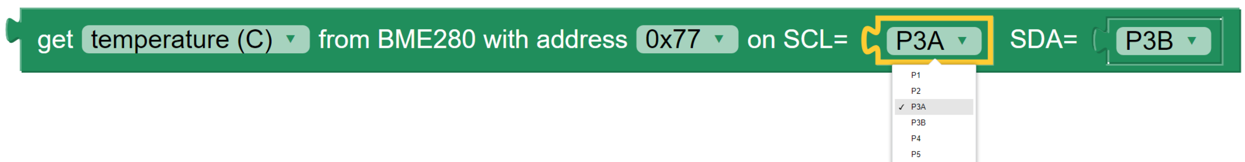
The first drop-down box provides the list of measurements available which are:

1. Temperature in degrees Centigrade
2. Air pressure in hectoPascals (aka milliBars)

3. Relative air humidity in percent
4. Altitude in metres relative to the first reading taken by the **KookaBlockly** script. That is, the first reading calibrates the altitude to zero metres.



The second drop-down box provides two options for the BME280's address on the I2C bus, that is `0x77` or `0x76`. The default of `0x77` is usually the best to use but it depends on what address the manufacturer of the BME280 sensor board has chosen. It is possible to have two BME280 sensors, each with a different address, on the same **Kookaberry** interface.



The third and fourth drop-down boxes provide options as to which **Pins** are used for the SCL and SDA signals on the **Kookaberry**.

Usually the defaults of `P3A` for SCL and `P3B` for SDA will work, using the **Kookaberry**'s **P3** 4-wire connector.

Some BME280 boards on the market have the SCL and SDA wires swapped, which requires the selections on the block to be swapped.

Any other of the **Kookaberry**'s connectors (**P1** to **P5**) can also be used.

A string block can also be used instead of the drop-down selector blocks and the name of the **Pin** typed into the block.

About The BME280 Sensor

The BME280 sensor measures air temperature, relative humidity, and barometric air pressure.

There is also a compatible BMP280 sensor that measures air temperature and barometric air pressure, but does not measure relative humidity. Using the blocks below will return a reading of zero for humidity.

This sophisticated sensor is available mounted on **Kookaberry**-compatible circuit boards distributed by a variety of manufacturers.

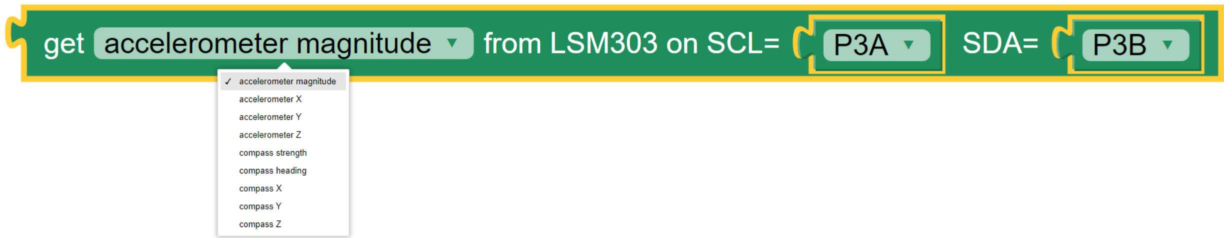
The interface with the **Kookaberry** is the I2C serial communications bus. I2C stands for Inter-Integrated-Circuit Communications (IIC or I2C). See https://en.wikipedia.org/wiki/I%C2%B2C for more detail.

There are four wires in the I2C interface, being: * `Vcc` power at +3.3 volts DC * `Gnd` ground (or negative) for signal and power at 0 volts * `SCL` being the serial clock signal for communications timing * `SDA` being the serial data signal which conveys the digital data being communicated

When using BME280 circuit boards it is important that these signals are connected to the correct **Pins** on the **Kookaberry**.

Get Acceleration / Compass Strength from LSM303

The **Get Acceleration from LSM303** block is shown below with the three sets of options available from the drop-down boxes on the block.



The first drop-down box provides the list of measurements available which are:

1. Acceleration total magnitude in metres / second squared
2. X axis acceleration in metres / second squared
3. Y axis acceleration in metres / second squared
4. Z axis acceleration in metres / second squared
5. Compass total magnetic field strength in Gauss
6. Compass heading in degrees from North
7. Magnetic field strength along the X axis in Gauss
8. Magnetic field strength along the Y axis in Gauss
9. Magnetic field strength along the Z axis in Gauss

The second and third drop-down boxes provide options as to which **Pins** are used for the SCL and SDA signals on the **Kookaberry**.

Usually the defaults of `P3A` for SCL and `P3B` for SDA will work, using the **Kookaberry**'s **P3** 4-wire connector.

Some LSM303 boards on the market have the SCL and SDA wires swapped, which requires the selections on the block to be swapped.

Any other of the **Kookaberry**'s connectors (**P1** to **P5**) can also be used.

A string block can also be used instead of the drop-down selector blocks and the name of the **Pin** typed into the block.

About the LSM303 Sensor

The LSM303 sensor contains a 3-axis accelerometer and a 3-axis magnetometer. The **Kookaberry** contains a LSM303 sensor internally, and this block provides functionality to use an externally connected LSM303 sensor.

This sensor can provide acceleration values and magnetic field strength in all three axes, total acceleration and total magnetic field strengths, as well as compass heading.

See https://www.explainthatstuff.com/accelerometers.html for a simple explanation of what an accelerometer is.

For an explanation of what a magnetometer is, see https://en.wikipedia.org/wiki/Magnetometer.

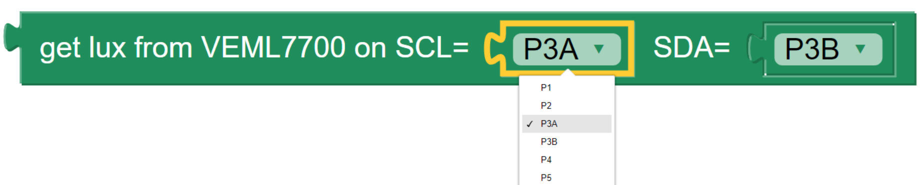
The interface with the **Kookaberry** is the I2C serial communications bus. I2C stands for Inter-Integrated-Circuit Communications (IIC or I2C). See https://en.wikipedia.org/wiki/I%C2%B2C for more detail.

There are four wires in the I2C interface, being: * `Vcc` power at +3.3 volts DC * `Gnd` ground (or negative) for signal and power at 0 volts * `SCL` being the serial clock signal for communications timing * `SDA` being the serial data signal which conveys the digital data being communicated

When using LSM303 circuit boards it is important that these signals are connected to the correct **Pins** on the **Kookaberry**.

Get LUX from VEML7700

The **Get Lux from VEML7700** block is shown below with the two sets of options available from the drop-down boxes on the block.



The two drop-down boxes provide options as to which **Pins** are used for the SCL and SDA signals on the **Kookaberry**.

Usually the defaults of `P3A` for SCL and `P3B` for SDA will work, using the **Kookaberry**'s **P3** 4-wire connector.

Some VEML7700 boards on the market have the SCL and SDA wires swapped, which requires the selections on the block to be swapped.

Any other of the **Kookaberry**'s connectors (**P1** to **P5**) can also be used.

A string block can also be used instead of the drop-down selector blocks and the name of the **Pin** typed into the block.

About the VEML7700 Sensor

The VEML7700 is a high-accuracy ambient light sensor with an I2C serial interface to the **Kookaberry**.

The ambient light readings are measured in Lux. Lux is the unit of illuminance, or luminous flux per unit area, in the International System of Units (SI), and is equal to one lumen per square metre. See https://en.wikipedia.org/wiki/Lux for more detail.

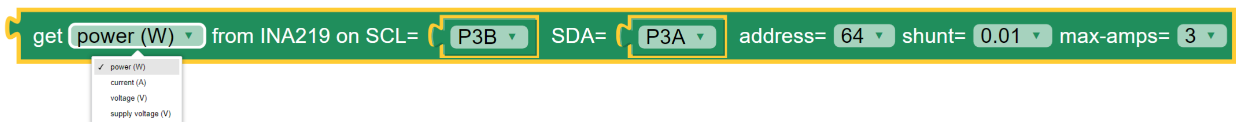
The interface with the **Kookaberry** is the I2C serial communications bus. I2C stands for Inter-Integrated-Circuit Communications (IIC or I2C). See https://en.wikipedia.org/wiki/I%C2%B2C for more detail.

There are four wires in the I2C interface, being: * `Vcc` power at +3.3 volts DC * `Gnd` ground (or negative) for signal and power at 0 volts * `SCL` being the serial clock signal for communications timing * `SDA` being the serial data signal which conveys the digital data being communicated

When using a VEML7700 circuit board it is important that these signals are connected to the correct **Pins** on the **Kookaberry**.

Get Power / Voltage / Current from INA219

The **Get Power / Voltage / Current from INA219** block is shown below with the four sets of options available from the drop-down boxes on the block.

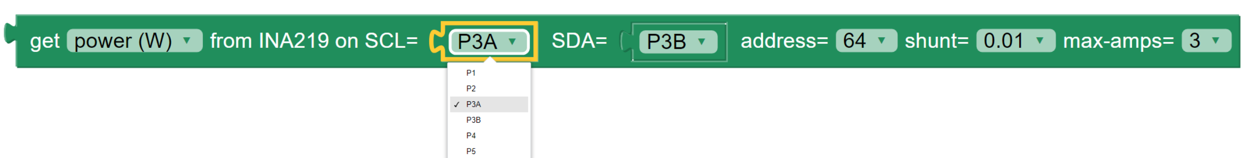


The first drop-down box provides the list of measurements available which are:

1. Power in watts DC (direct current).
2. Current in amperes (amps) DC.
3. Load voltage in volts DC.
4. Power supply voltage in volts DC.

Note: The range and resolution of the INA219 sensor readings are set by the value of an internal shunt resistor, the maximum amps, and the interfacing software.

Important: The safe operating range of the INA219 is given by the device's data sheet. Nominally the maximum voltage is 26 volts, maximum current is 8 amps.



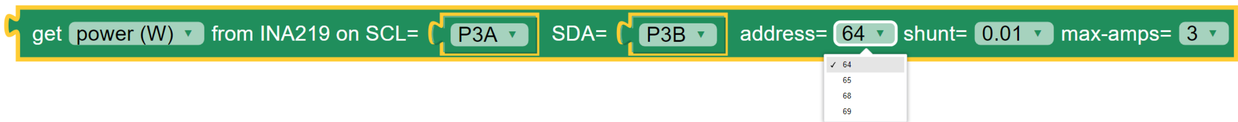
The second and third drop-down boxes provide options as to which **Pins** are used for the SCL and SDA signals on the **Kookaberry**.

Usually the defaults of `P3A` for SCL and `P3B` for SDA will work, using the **Kookaberry**'s **P3** 4-wire connector.

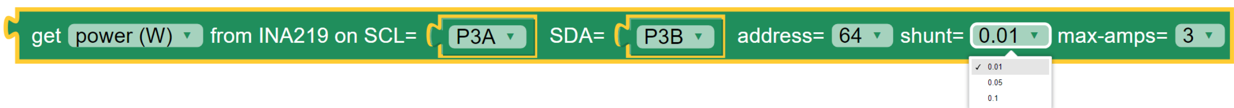
Some INA219 boards on the market may have the SCL and SDA wires swapped, which requires the selections on the block to be swapped.

Any other of the **Kookaberry**'s connectors (**P1** to **P5**) can also be used.

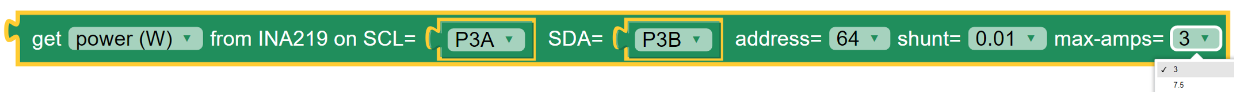
A string block can also be used instead of the drop-down selector blocks and the name of the **Pin** typed into the block.



The fourth option on the block is the I2C address of the board. Up to four INA219 sensors may be connected to a single I2C bus with any of the addresses `64` (hex `0x40`), `65` (hex `0x41`), `68` (hex `0x44`) or `69` (hex `0x45`). Each board must have a unique I2C address. To change the address in the block select the desired address from the drop-down list.



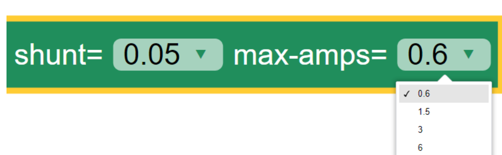
The fifth option is a drop-down list of shunt resistors fitted to the sensor. The correct value can be obtained by consulting the data sheet for the sensor board that is being used. This value must be set correctly or else erroneous readings will result. There are three options for shunt resistor values: `0.01` ohms, `0.05` ohms, and `0.1` ohms. Larger shunt resistance will improve the resolution of the current reading but will reduce the maximum current that can be measured. Care must also be taken to not exceed the shunt resistor's power rating which is typically 2 watts. Power in the shunt resistor is dissipated as heat and is equal to i^2 x R, where i is current in amps, and R the resistance in ohms.

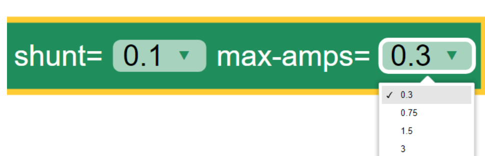


The sixth option is a drop-down list of the maximum currents to be measured. The values in he list change according to the shunt resistance selected.

To achieve the best resolution in current measurements, a the maximum current above and closest in value to the maximum current expected through the load should be selected. The block will try to optimise the INA219 sensor settings for a given shunt resistor and to avoid selecting currents which are beyond the safe operating range of the sensor.

The available combinations of shunt resistor and max-amps are shown below.





About the INA219 Sensor

The INA219 sensor measures direct current, voltage and power from the circuit to which it is connected. It is commonly called a wattmeter.

In a direct current circuit, electrical power delivered to an electrical load (measured in watts) is the arithmetic product of the current flowing through the load (measured in amperes) and the voltage across the load's terminals (measured in volts).

To measure the current, a low value resistor is placed in series with the load, and the voltage across the resistor's terminal is measured. By applying Ohm's Law, the current can be derived (current I = voltage V / resistance R).

See also

- https://en.wikipedia.org/wiki/Voltmeter,
- https://en.wikipedia.org/wiki/Ammeter and
- https://en.wikipedia.org/wiki/Ohm%27s_law

The INA219 sensor is commonly mounted on a breakout board equipped with terminals to attach the load and a power supply, and a shunt resistor used to measure current flowing through the load.

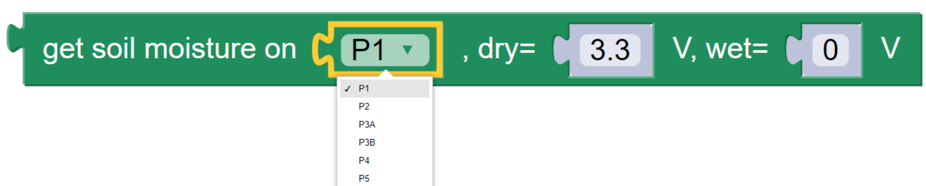
The interface with the **Kookaberry** is the I2C serial communications bus. I2C stands for Inter-Integrated-Circuit Communications (IIC or I2C). See https://en.wikipedia.org/wiki/I%C2%B2C for more detail.

There are four wires in the I2C interface, being: * `Vcc` power at +3.3 volts DC * `Gnd` ground (or negative) for signal and power at 0 volts * `SCL` being the serial clock signal for communications timing * `SDA` being the serial data signal which conveys the digital data being communicated

When using a INA219 circuit board it is important that these signals are connected to the correct **Pins** on the **Kookaberry**.

Get Soil Moisture

The **Get Soil Moisture** block is shown below with three options available on the block.



Soil moisture is given as a percentage, nominally in the range 0 to 100. Values outside that range can be returned depending on the calibration values set in the `dry=` and `wet=` fields on the block.

The first option is a drop-down block to select which **Pin** the sensor is connected to. A **String** block can also be used instead of the drop-down selector block and the name of the **Pin** typed into the block.

To the right of the **Pin** selector drop-down list are two fields which can be manually edited. These are the voltages given by the sensor when it is dry and when it is wet. The default values suit a capacitive sensor.

1. For a resistive sensor, the dry value should be lower than the wet value. `Dry=` 0 volts and `wet=` 3.3 volts are suitable starting values.
2. For a capacitive sensor, the dry value should be higher than the wet value. `Dry=` 3.3 volts and `wet=` 0 volts are suitable starting values.

These values can be tuned with experience and the use of a calibrated soil moisture meter to improve the accuracy of the readings.

About Soil Moisture Sensors

There are two types of soil moisture sensor available:

1. Resistive soil moisture sensor which measures the conductivity of soil by applying an electrical voltage using two spikes.
2. Capacitive soil moisture sensor, consisting of a single broad spike, which measures changes in the soil's capacitance due to the presence of moisture.

While both kinds of sensor are effective, the capacitive soil moisture sensor is much more durable as it is not susceptible to corrosion which affects resistive sensors in prolonged use.

Learn more about using the resistive soil moisture sensor here: https://learn.auststem.com.au/peripheral/analogue-soil-moisture-sensor/

2.7.3 More Sensor Learning Resources

More information on sensors that can be used with the **Kookaberry** is here: https://learn.auststem.com.au/peripherals/

2.8 Actuators

The Actuators category provides the blocks that enable the use of these servos. See Fig. 2.19.

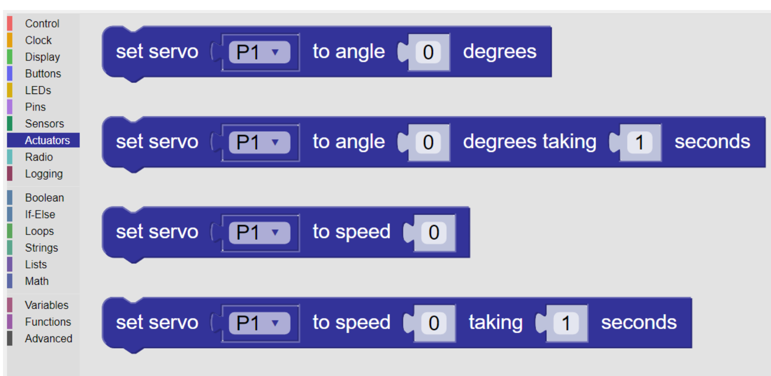


Fig. 2.19: The palette of **KookaBlockly** Actuator blocks

The Actuators category comprises blocks to use Hobby Servos and Continuous Rotation Servo Motors.

Hobby Servos and have a built in motor, a feedback circuit and a motor driver. They can be set to a particular angle and have a constrained range of motion, typically 180 degrees. These servos are used in robot arms, for example.

Continuous Rotation Servos, as the name implies, can rotate continuously like a motor. The control signal sets the speed of rotation, typically in degrees per second. Continuous rotation servos can be used for driving the driving wheels of vehicles.

The supported servo motors have a three pin connector comprising:

a. Gnd - power supply ground

b. Vcc - positive DC power supply, and

c. A pulse servo signal that controls the servo motion.

A typical Hobby Servo operates with a power supply voltage of around 4.5 to 6 volts.

While it is possible to drive some small servos directly from the **Kookaberry**, it is recommended that the servo be powered a separate power supply due to the required servo power being higher than the **Kookaberry** can provide. A directly connected servo will be weak and slow, and may result in the **Kookaberry**'s power supply shutting down on overload.

2.8.1 Actuators' Pins Connections

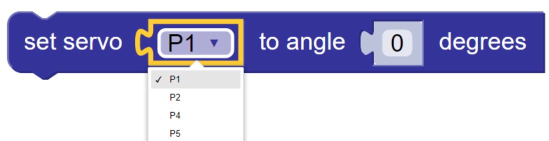
Actuators are connected to the **Kookaberry** by one of the five connectors on the back, `P1` through to `P5`, with connector `P3` having two possible connection points: `P3A` and `P3B`. (see the *Pins* category description).

Each actuator block has an input **Pins** drop-down selection blocks by which the input Pin can be selected.

It is possible to replace the **Pins** dropdown selection block with a **String** block. This is useful when using **Pins** other than those exposed on the rear of the **Kookaberry**, or when other microprocessor boards that are compatible with **Kookaberry** firmware are being used. In those cases type in the Pin's identifier into the **String** block.

2.8.2 Set Servo to Angle

This block is for a Hobby Servo, which is a servo is a motor that rotates over a specified angular range.



The servo block sets the angle to which a servo motor should move specified in degrees. The angle can be calculated by other value blocks or be specified as a fixed value. The option for this block is which connector the servo is attached.

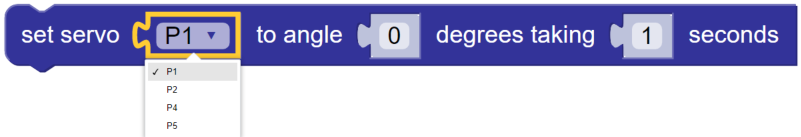
The block has two parameters:

1. A dropdown block to selected which Pin the servo's control signal is connected to. A string block can also be used instead of the drop-down selector blocks and the name of the Pin typed into the block.
2. The angle, in degrees, to which the servo is to rotate. The angle can be between - (range of rotation) / 2 to + (range of rotation) / 2. The rotation will occur almost instantly.

Important: Please note that all but the smallest 9g servos should not be directly plugged into a **Kookaberry** connector. These devices require special electronics to supply them with more power. Plugging in large servos without the necessary driving electronics may shut down and possibly irreparably damage the **Kookaberry**!

2.8.3 Set Servo to Angle Taking Seconds

This block is the same as the **Set Servo to Angle** block with the addition of a parameter to set the time, in seconds, over which the angular motion should occur. This allows for a less abrupt and more graceful motion of the servo.

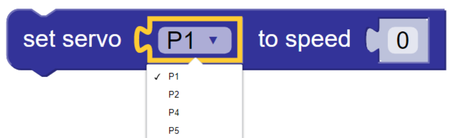


The block has three parameters:

1. A dropdown block to select which Pin the servo's control signal is connected to. A string block can also be used instead of the drop-down selector block and the name of the Pin typed into the block.
2. The angle, in degrees, to which the servo is to rotate.
3. The time, in seconds, over which the rotation will occur.

2.8.4 Set Servo to Speed

This block is for a Continuous Servo, which is a motor that rotates at a specified rotational speed.



The servo block sets the angular speed at which a servo motor should rotate specified in degrees per second. The speed can be calculated by other value blocks or be specified as a fixed value. The option for this block is which connector the servo is attached.

The block has two parameters:

1. A dropdown block to select which Pin the servo's control signal is connected to. A string block can also be used instead of the drop-down selector block and the name of the Pin typed into the block.
2. The speed at which the servo is to rotate in degrees / second. The target speed will occur almost instantly.

Important: Please note that all but the smallest 9g servos should not be directly plugged into a **Kookaberry** connector. These devices require special electronics to supply them with more power. Plugging in large servos without the necessary driving electronics may shut down and possibly irreparably damage the **Kookaberry**!

2.8.5 Set Servo to Speed Taking Seconds

This block is for a Continuous Servo, which is a motor that rotates at a specified rotational speed.

This block is the same as the **Set Servo to Speed** block with the addition of a parameter to set the time, in seconds, over which the change in angular speed should occur. This allows for a less abrupt and more graceful transition in the speed of the servo.

The block has three parameters:

1. A dropdown block to select which Pin the servo's control signal is connected to. A string block can also be used instead of the drop-down selector block and the name of the Pin typed into the block.
2. The speed, in degrees / second, at which the servo is to rotate.
3. The time, in seconds, over which change to target speed will occur.

2.8.6 More Actuator Learning Resources

More information on using actuators with the **Kookaberry** can be found here: https://learn.auststem.com.au/peripheral/micro-servo/

2.9 Radio

Radio communications between **Kookaberries** is possible using the **Radio** blocks shown in Fig. 2.20.

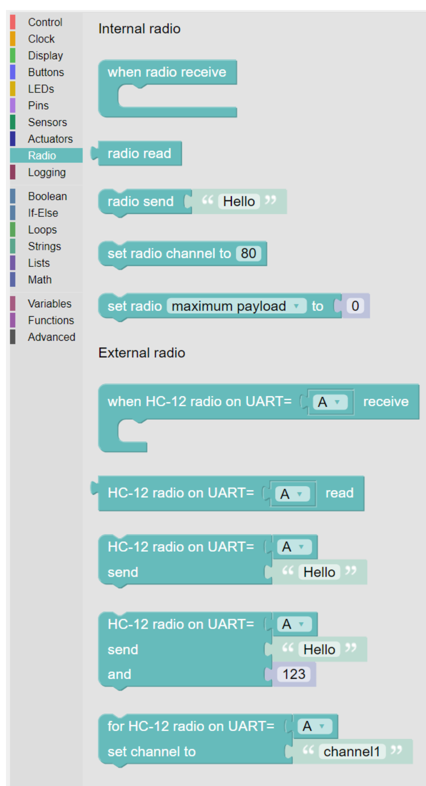


Fig. 2.20: The palette of **KookaBlockly Radio** blocks

Radio communications is useful for sending messages, sharing data, for remote monitoring, and for remote control.

The **Kookaberry** has an internal short-range digital packet radio, and can also connect to one or more external longer range radios.

2.9.1 Internal Radio

The **Kookaberry** is equipped with a built-in digital radio transceiver than is able to send and receive small amounts of digital data.

The radio uses the same radio spectrum as WiFi signals and Bluetooth signals, and therefore has a similar range of 10 to 20 metres. The internal radio cannot communicate using WiFi or Bluetooth directly.

All **Kookaberries** on the same radio channel can listen in to the communications on that channel.

Similarly, multiple **Kookaberries** transmitting on the same channel may interfere with each others' communications. Errors caused during radio communications are detected and messages with errors caused by interference will be discarded.

The internal radio is compatible with the **BBC Micro:Bit**'s radio, as it uses the same radio chip, radio frequencies, and digital signalling.

It is possible to exchange messages between the **Kookaberry** and the **Micro:Bit** provided the same radio channel is selected on both devices, nominally on Channel 7.

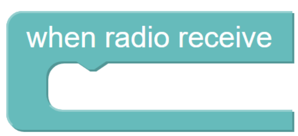
By default, the length of the messages that can be sent is 30 bytes or less when using **KookaBlockly**. Other **Radio** parameters such as the radio channel and speed of transmission are also set to default values.

In the latest release of **KookaBlockly**, functionality has been added to alter the default parameters of the internal radio. Care must be taken however that all the **Kookaberries** involved in communication have their radio parameters set in the same way.

The following blocks are available to control, receive and send messages using the internal **Kookaberry** radio.

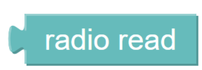
When Radio Receive

This is a control block which contains actions that will be taken when a message is received by the **Radio**. If no message is received then no actions within the scope of the block will be taken.



Radio Read

This value block will read the first **Radio** message in the queue of **Radio** messages received. Once read the **Radio** message is deleted from the message queue.



Radio Send

This action block sends the data within the attached value block as a message via the **Radio** to be received by all other radios on the same channel.

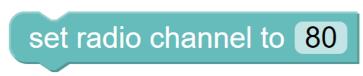
The data can be the result of a value block, or be a fixed message as shown above.

The length of the message must be no longer than the message length limit or else a program error will result.

Typically an alphanumeric text character occupies only one byte but some special characters may occupy two or more bytes.

Set Radio channel

This block enables any of the available **Radio** channels to be selected.



The **Kookaberry**'s internal radio is capable of transmitting and receiving on any of 84 channels. The default **Radio** channel is 7.

An integer value between 0 and 83 can be selected by editing the number in the block.

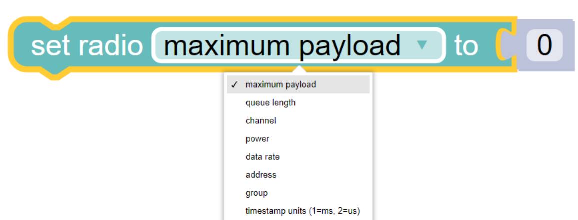
Messages will be sent via this channel and only messages received via this channel will be put onto the incoming message queue.

It is therefore important that for two or more **Kookaberries** to intercommunicate, that they all be set to the same channel.

Each channel is 1MHz wide, starting at Channel 0 at 2400MHz and ending at Channel 83 at 2483MHz.

Set Radio Parameter

The **Kookaberry**'s internal radio can be configured in a variety of ways if the default settings are not suitable.



This block provides access to the numerous parameters that can be set.

Only one parameter can be set per instance of the block. Multiple instances of the block must be used to set multiple **Radio** parameters.

The block contains a drop-down list that enables selection of which parameter is to be set, and an input for a block that specifies the value of the selected parameter:

1. `maximum payload` (default=32) defines the maximum length, in bytes, of a message sent via the **Radio**. It can be between 1 and 251 bytes long.

2. `queue length` (default=3) specifies the number of messages that can be stored on the incoming message queue. If there is no space left on the queue then additional incoming messages are dropped. Can be between 1 and 254.
3. `channel` (default=7) an integer value between 0 and 83 inclusive that defines the channel (actually frequency) to which the **Radio** is tuned. Messages will be sent via this channel and only messages received via this channel will be put onto the incoming message queue. Each step is 1MHz wide, starting at 2400MHz.
4. `power` (default=6) an integer value between 0 and 7 inclusive which indicates the strength of signal used when sending a message. The higher the value the stronger the signal, but the more power is consumed by the device. The numbering translates to positions in the following list of dBm (decibel milliwatt) values: -30, -20, -16, -12, -8, -4, 0, 4.
5. `data rate` (default=1) indicates the speed at which data transfer (send and receive) takes place. It can be 0, 1 or 2, for 250kbit/sec, 1Mbit/sec, or 2Mbit/sec respectively
6. `address` (default=0x75626974) an arbitrary name, expressed as a 32-bit address, that's used to filter incoming packets at the hardware level, keeping only those that match the address you set. The default matches that used on the micro:bit.
7. `group` (default=0) an 8-bit value (0-255) used in conjunction with address to filter incoming messages. This effectively makes the full address 40 bits long.
8. `timestamp units` (default=1) an integer 1 (TIMESTAMP_MS milliseconds) or 2 (TIMESTAMP_US microseconds) that indicates the units used in the timestamp entry returned by the receive_full() function.

Note: It would be very unusual to alter any of the **Radio** parameters, other than the channel, when coding using **KookaBlockly**.

2.9.2 External Radio

The **Kookaberry** can be connected to up to two external radio transceivers to communicate with other **Kookaberries** (or other computers) that use the same radio transceivers.

The preferred radio transceiver is the HC-12 transceiver which operates in the 433Mhz radio band.

This radio band is the same as is used for domestic applications such as garage door openers and home weather stations. It offers the advantage of communicating over a longer range than the **Kookaberry**'s internal radio.

Depending on the antenna fitted and the intervening radio environment, a range of at least 100 metres can be expected, with up to 1 kilometre possible in the right circumstances.

Successful communication requires that all transceivers are set to the same parameters, particularly the same radio channel.

Setting up the HC-12 to other than its default parameters is beyond the scope of **KookaBlockly**. Please refer to the HC-12 data sheet at https://www.elecrow.com/download/HC-12.pdf.

Radios other than the HC-12 can be used provided they emulate a wired connection and do not require any control commands.

The interface to the **Kookaberry** is via its UART (Universal Asynchronous Receiver and Transmitter) serial interface at 9600 bits/second.

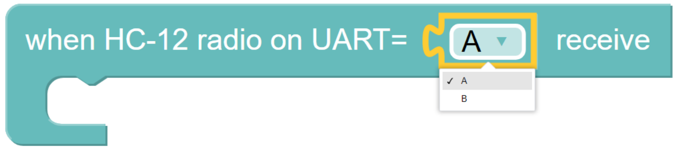
Two UART interfaces are available on the **Kookaberry**:

A. This interface is accessed by using plug `P3` on the back of the **Kookaberry**. This is **Radio** `A`.

B. This interface requires an expansion board that connects via the **Kookaberry**'s edge connector. The plug on such a board is `P6`. This **Radio** is designated **Radio** `B`.

When HC-12 Receive

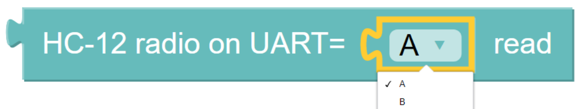
This is a control block which contains actions that will be taken when a message is received by the selected external radio. If no message is received then no actions within the scope of the block will be taken.



The drop-down list on the block selects which of the external radios (`A` or `B`) is being used.

HC-12 Read

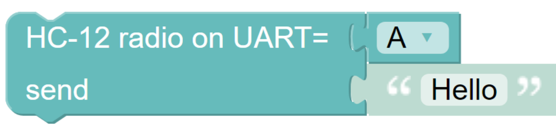
This value block will read the first **Radio** message in the queue of **Radio** messages received by the external radio. Once read the **Radio** message is deleted from the message queue.



The drop-down list on the block selects which of the external radios (`A` or `B`) is being used.

HC-12 Send

This action block sends the data within the attached value block as a message via the external radio to be received by all other radios on the same channel.

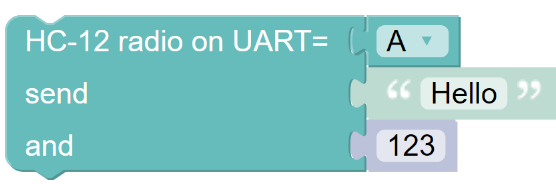


The data can be the result of a value block, or be a fixed message as shown above.

The drop-down list on the block selects which of the external radios (`A` or `B`) is being used.

HC-12 Send and

This action block sends the data within the attached value blocks as a message via the external radio to be received by all other radios on the same channel.



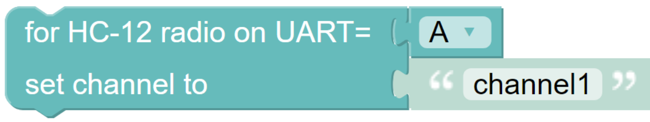
The data sent is a concatenation of the two value blocks.

The first block can be a descriptor (eg. **Temperature**) and the second the value derived from a temperature sensor.

The drop-down list on the block selects which of the external radios (A or B) is being used.

HC-12 Set Channel

This block sets a virtual (named) channel for the external radio.



The external radio will send all messages with a prefix equal to the channel name.

The external radio will also only receive messages with the same channel name.

Note: This virtual channel does not affect the radio frequency that the external radio uses. It is only a prefix that groups messages into groups.

The drop-down list on the block selects which of the external radios (A or B) is being used.

2.10 Logging

The **Logging** blocks, shown in Fig. 2.21, provide a facility for writing data into files on the **Kookaberry**.

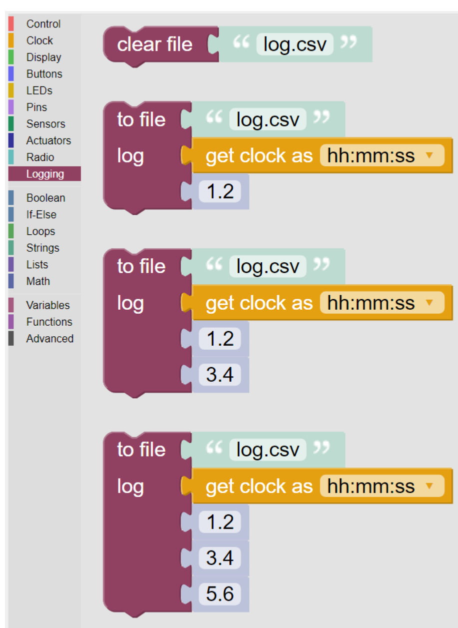


Fig. 2.21: The palette of **KookaBlockly Logging** blocks

Note: At present **KookaBlockly** does not directly support the reading of files from the **Kookaberry**'s file system.

MicroPython scripting does however contain extensive functionality for reading, writing and manipulating the **Kookaberry**'s files. In the *Advanced* Category there is an example of using **Python** blocks to read a text file.

The **Kookaberry** contains a 3 to 4 megabyte (depending on hardware model) non-volatile serial memory store which is used to store files. These files can be written and read by the **Kookaberry** and also via a USB interface by any attached computer.

Logging files are text files which are in the comma-separated-values (**CSV**) format. That is, each line contains alphanumeric text data which are separated by commas. The first line of the files can be used to represent headings for the data item columns that are in the following lines. An example of a **CSV** file is:

```
Time,Temperature,Humidity
12:04:00,25,50
12:09:00,26,49
12:14:00,27,48
etc
```

During experiments, data is collected over time from instruments comprising sensors. These data are stored in a **CSV** file at time intervals as above.

When the experiment is finished, the data can be retrieved from the **CSV** file stored on the **Kookaberry** using a computer to perform analysis of the results. **CSV** text data is most commonly used to draw graphs of the data values over time using a spreadsheet program.

2.10.1 Clear File

The file block creates a new empty text file with the specified name in the **Kookaberry**'s file system. If a file with the same name already exists, then it will overwritten with an empty file.

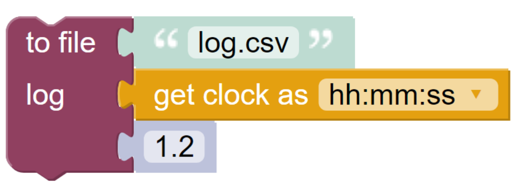
The name of the file is specified in the **to file** parameter with `log.csv` the default name. Edit this field to change the file name. This can be any legal filename, usually in the form `name.typ` where name is a text string and typ is a short, usually three letter, file type description.

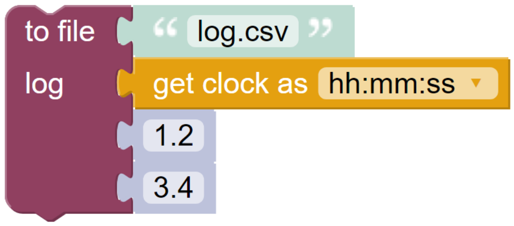
`CSV` is the recommended file type, but other common types are: `txt` for text files, and `log` for log text files. File type conventions are determined by the computer operating system that will read these files.

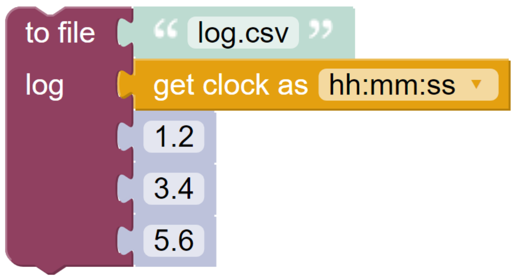
2.10.2 Log To File

The **Log To File** block writes the text provided by the attached value block(s) as a new line appended to the named text file. If the text file name does not already exist, a new empty text file with the specified name will be created.

The value blocks attached as inputs to this block will provide text values to be written to the line in the file, separated by commas.







The first input, by default, is a text representation of the current time read from the **Kookaberry**'s internal clock. This input block can be replaced by any other value block that provides a text string.

There are three varieties of the **Log To File** block, accepting one two or three further inputs. These inputs are also expected to be text string representations of the data to be recorded in the file record.

To create a heading line in the **CSV** file, use the appropriate **Log To File** block first within an **On Start** control block and plug in text string value blocks with the names of each of the columns.

Note: **KookaBlockly** presently supports a maximum of four data items per file record inclusive of the time string input.

If logging the time is not needed, then the time string can be replaced with some other string input.

If more data items are required then it is possible to use an *Advanced* block with the required MicroPython script in it.

The **Show Script** button on the **KookaBlockly** editor will open a window with the MicroPython script derived from the current **KookaBlockly** script.

Hint: Use a **Log To File** block to model the first four data items, copy the equivalent MicroPython (it all has to be on one line), paste it into the *Advanced* block and modify it to suit your application.

You will need to learn about MicroPython nonetheless to make it work correctly.

2.11 Boolean

Boolean blocks are value blocks used to test whether a specified condition is `True` (1) or `False` (0). See Fig. 2.22.

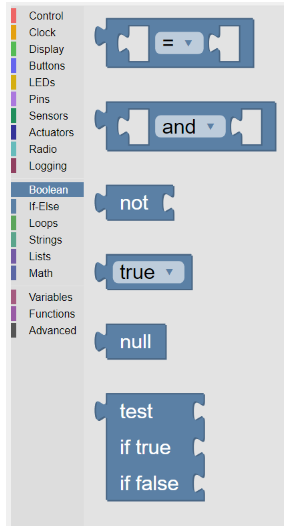
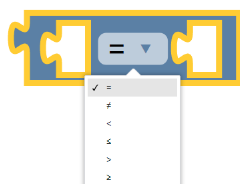


Fig. 2.22: The palette of **KookaBlockly Boolean** blocks

2.11.1 Comparison

This Comparison block compares the two value blocks that are given with the rule selected from the dropdown menu and outputs a result of `True` or `False`.



The options available in the drop-down selection box are:

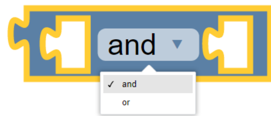
1. the inputs are equal (=)
2. the inputs are not equal ()
3. the first input is less than (<) the second input
4. the first input is less than or equal to () the second input
5. the first inputs is greater than (>) the second input
6. the first input is greater than or equal to () the second input.

Equal to (=) and not equal to () work for almost anything including numbers, lists (arrays) and character strings.

The other operands only work for numbers.

2.11.2 Boolean And / Or

The **Boolean And / Or** block performs the selected **Boolean** operation on its two inputs.



Both inputs are required to be **Boolean**. It is not possible to plug numbers or text strings into the inputs.

- `and` will give back a `True` only if both of its inputs are `True`.
- `or` will give back `True` if either or both of its inputs are `True`.

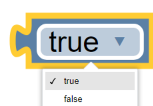
2.11.3 Not

This block takes a `True/False` **Boolean** value block input and logically inverts it.

That is, `True` becomes `False`, and `False` become `True`.

2.11.4 True / False

This value block gives a **Boolean** `True` or `False` value depending on which option is selected. It is generally used to initialise variables that are subsequently used in a program.

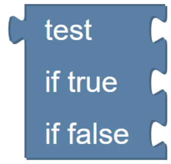


2.11.5 Null

This value block is the value that variables have before they are given a value. It is a special value that represents "none" or "nothing" but is distinct from 0. However it is treated as a zero or `False` value if used.

2.11.6 Test If

This block will output one of two input values depending on whether the **test** input is `True` or `False`.



If the block in the **Test** input socket is `True`, the value in the **if true** input is transferred to the output.

If the block in the **Test** input socket is `False`, the value in the **if false** input is transferred to the output.

2.12 If–Else

The **If-Else** category comprises control blocks which direct the flow of a program depending on the results of the tests carried out by these blocks. See Fig. 2.23.

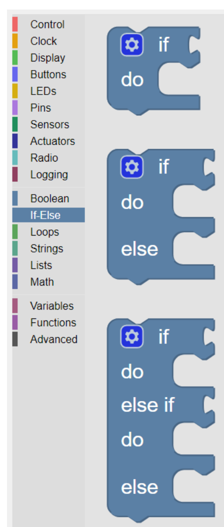
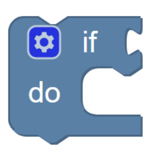


Fig. 2.23: The palette of **KookaBlockly If-Else** blocks

2.12.1 If-Do

The **if** input socket takes a value block or compound block that represents a `True` or `False` value.



If the value block in the conditional input is `True`, it runs the blocks nested inside.

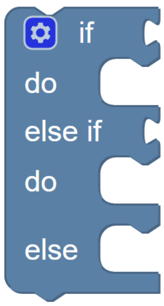
If the block in the conditional input is False, it skips the nested blocks.

2.12.2 If-Do-Else-Do

This block is an extension of the **If-Else-Do** block. It adds the **else** bracket into which the action blocks that are to be run if the tested input is `False`.

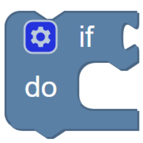
2.12.3 If-Do-Else If-Do-Else-Do

This block is a further extension of the **If-Do-Else-Do** block. A second conditional **else if** input is inserted and a bracket for actions to be run if the **else if** input is `True`.



2.12.4 If-Do Configuration

The **If-Do** block is configurable.



By clicking the gear icon on the block, extra elections can be added by dragging the **else if** or **else** blocks into the white area to connect under the **if** block in the configuration box:

- **else if** sections can add more conditional sockets to check for further input **Boolean** values, and a **do** bracket to contain action blocks to be run if the input is `True`. Multiple **else if** sections can be configured.

- a single **else** section can be added to the end to contain the action blocks to be run if none of the previous conditions are `True`.

To remove any of the **else if** or **else** sections, drag them back into the grey area of the configuration box.

To close the configuration box, simply click the gear icon once more.

2.13 Loops

Loops are a category of control blocks, shown in Fig. 2.24, that direct the flow of a program. They run the nested action blocks a number of times in accordance with the test taken at the beginning of the **Loop**.

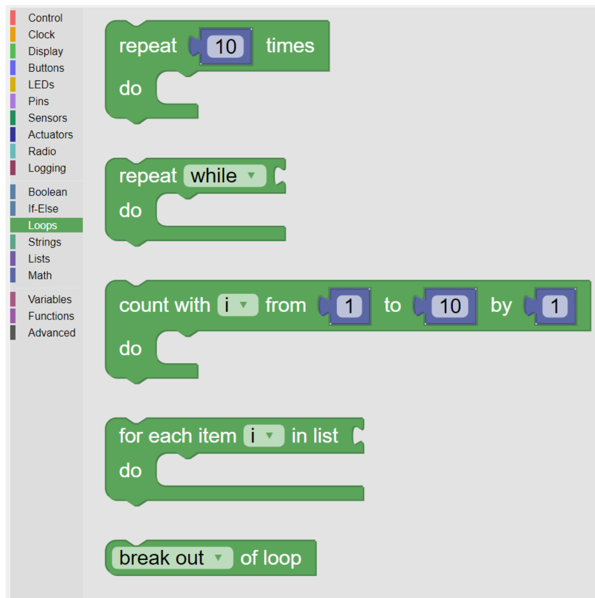
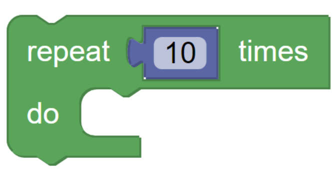


Fig. 2.24: The palette of **KookaBlockly Loop** blocks

2.13.1 Loop Repeat

This block runs the blocks nested inside of it for the specified number of times.

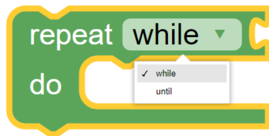


The number of iterations is provided by an input from a numeric value block which can contain a fixed number (from the *Math* blocks category), a numeric computation (using blocks from the *Math* blocks category), or a variable. See also the *Variables* category.

When the iterations of the **Loop** are complete the program moves on to the blocks below it.

2.13.2 Loop Repeat While / Until

In this block the two operations of While and Until are very similar to each other. Both require a *Boolean* `True / False` value block in their input socket.

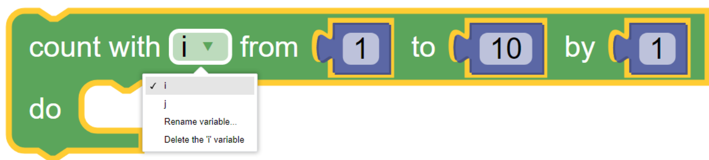


Repeat While will continue as long as the input value block is `True`.

Repeat Until will continue as long as the input value block is `False`.

2.13.3 Count With Variable From-To-By

This **Loop** will run its nested blocks several times depending upon the input numbers given.



The **Loop** will start by setting the chosen variable to its starting value using the first input.

Each time the **Loop** completes (known as an iteration), the variable's value is changed by the number in the third input.

The **Loop** will continue to iterate until the value of the variable is equal to or greater than the number in the second input.

So if the **Loop** is configured to run from `0` to `3` by `1`, it would run the nested blocks with the variable's value being `0`, `1` and `2`. Then the program would advance to the next block after the **Loop**. During the **Loop**, the variable's value indicates which repetition of the **Loop** is being run and can be used in calculations.

The variable drop-down list contains the names of the available variables. The default variables are `i` and `j`.

The options **Rename variable** and **Delete variable** are configuration functions to manage the creation of new variables or deletion of existing variables. See also the *Variables* Category.

Count With Variable Example

In Fig. 2.25 is an example of the **Loop** counting between 1 and 16 by 3.

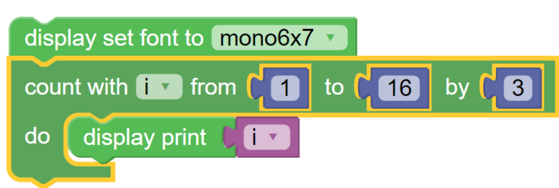


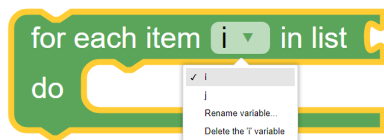
Fig. 2.25: Example script counts from 1 to 16

On each iteration of the **Loop**, the value of the variable `i` is printed on a new line on the display, as shown in Fig. 2.26.

Fig. 2.26: The display resulting from Fig. 2.25

2.13.4 For Each Item In List

This block has an input socket that takes a **List**. See the *Lists* Category.



The **Loop** begins by setting the chosen variable to be the same as the first item from the **List** and then it runs the nested blocks.

The **Loop** then sets the chosen variable as the second item of the **List** and runs the nested blocks again.

The **Loop** repeats until it has run once for every item from the **List**.

This type of **Loop** is useful for printing a **List** of text items in subsequent lines on the **Display**, or for processing a **List** of readings gathered from sensors.

2.13.5 Break / Continue Loop

This block must be placed inside a **Loop**. If the block is not placed in a **Loop** it will turn white with a warning symbol - see Fig. 2.28.

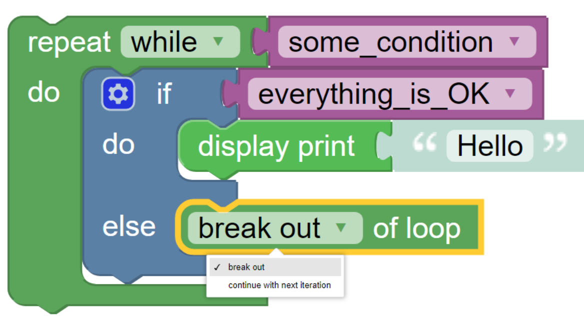


Fig. 2.27: The Loop Breakout / Continue used in a **Loop**

This block is used to either break out of the **Loop**, or to stop the current iteration of a **Loop**.

- **break out** immediately ends the **Loop** and jumps to the next block after the **Loop**.
- **continue with next iteration** stops the current iteration and jumps back to the top of the **Loop** and will run again if the **Loop** allows it.

The usual way to use this block is in an **If-Do** block where breaking a **Loop** is subject to a logical test as in Fig. 2.27.

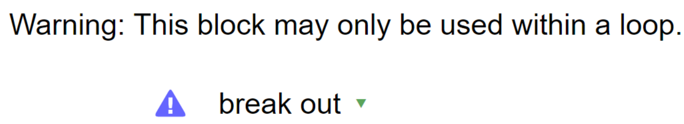


Fig. 2.28: The Warning appearance of the **Loop Breakout / Continue** block when not inside a **Loop**

2.14 Strings

Strings are an array of consecutive text characters such as "`Hello`", or "`this is a string`".

The **Strings** Category provides a set of value blocks for specifying and formatting strings, as shown in Fig. 2.29.

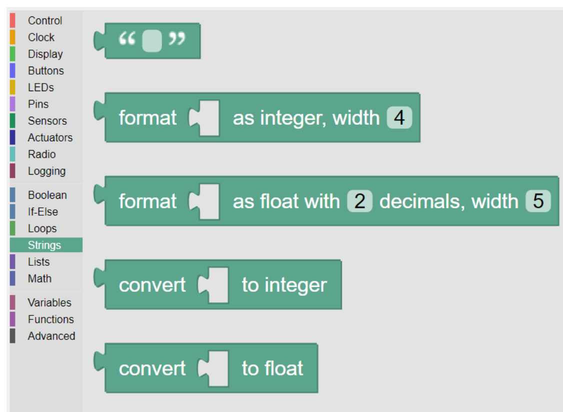


Fig. 2.29: The palette of **KookaBlockly String** blocks

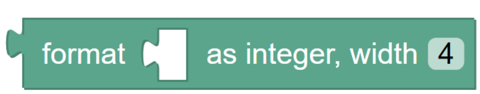
2.14.1 Text

This block allows a user to type in text that can be used as a string value by other blocks.

Type in the desired text between the double-quotes ", for example "`Hello World`".

2.14.2 Format as Integer

This block takes a numerical value block and formats its result as an integer with a width as defined in the block.



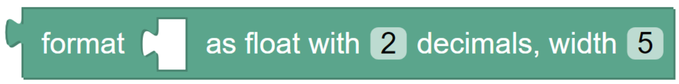
For example, the integer `1000` would be formatted as the character string "`1000`".

The results will in some cases vary:

- if the integer is wider than the specified width, the format will be enlarged to accommodate the number of characters required. For example, if the width is specified as `2` but the number is `1000`, the output will have width of `4` being `"1000"`.
- if the specified width of the output is greater than the width required, then leading spaces will be added. For example, if the width is specified as `2` but the number is 4, the output will be `"4"`.

2.14.3 Format as Floating Point

This block takes a numerical value block and formats its result as a floating point number with the specified number of decimal places and width (not including the decimal point).



For example, the number 123.4567 formatted as 2 decimals with width 5, would result in the character string "123.46". Note that the last digit is rounded up if greater than or equal to 5 or down if less than 5.

The results will in some cases vary:

- if the number is wider than the specified width, the format will be enlarged to accommodate the number of characters required. For example, if the width is specified as `3.2` but the number is `1000.12`, the output will have width of `6.2` being `"1000.12"`.
- if the specified width of the output is greater than the width required, then leading spaces and trailing zeroes will be added. For example, if the width is specified as `4.2` but the number is `3.1`, the output will be `" 3.10"`.

2.14.4 Convert to Integer

This block converts an input string value and outputs a numeric integer value.

For example, an input of `"1234"` will output the integer number `1234`.

Inputs strings that are not numeric integers, for example `"ten"` or `"10.1"`, will raise a formatting error and the script will terminate.

Numeric inputs are permitted, for example a floating point input `10.1` will yield an integer output `10`. Integer inputs will be passed through as integer outputs.

This block is useful when parsing text from the *Radio* into integer data for use in computations.

2.14.5 Convert to Float

This block converts an input string value and outputs a numeric floating point value.

For example, an input of `"1234.56"` will output the integer number `1234.56`.

Inputs strings that are not numeric floats, for example `"ten point one"` will raise a formatting error and the script will terminate.

Numeric inputs are permitted, for example an integer input `10` will yield an integer output `10.0`. Floating point inputs will be passed through as floating point outputs.

This block is useful when parsing text from the *Radio* into floating point data for use in computations.

2.15 Lists

The **Lists** category, shown in Fig. 2.30, provides a large number of blocks to create and manipulate **Lists**.

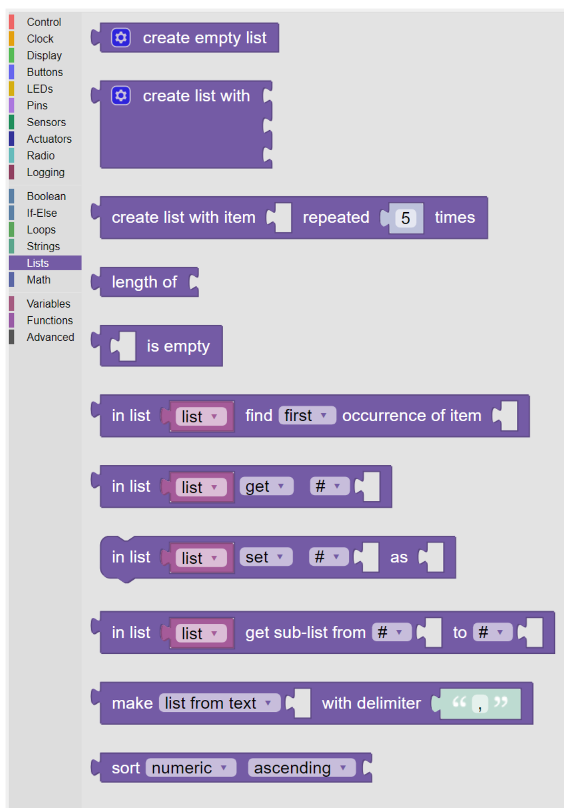


Fig. 2.30: The palette of **KookaBlockly List** blocks

A **List** is an array of zero or more items which can be *Variables*, numbers, characters, text, or other **Lists**.

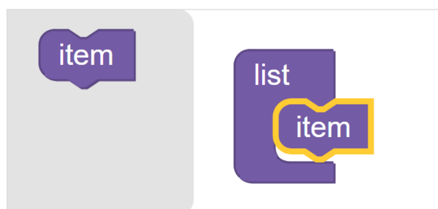
To create a **List**, first create a **Variable** with the name of the **List**, and then set its value to that returned by the **Create List** block.

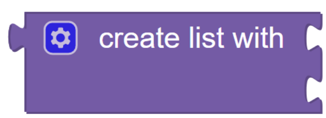
See the *Variables* Category to learn about creating and using **Variables**.

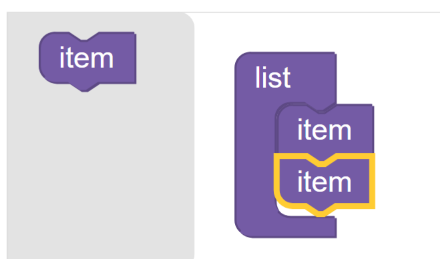
2.15.1 Create List

This value block gives back a new, empty **List**.

The gear icon in the block allows the user custom tailor the block to add items.

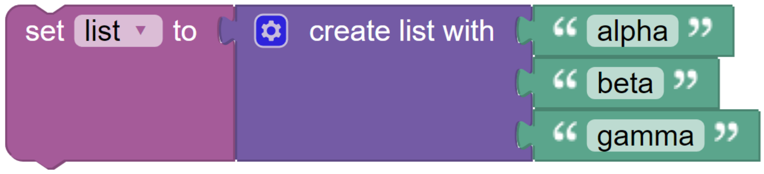






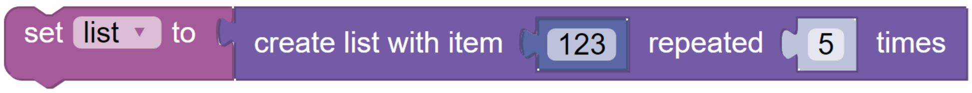
Create List Example

Here is an example of setting the value of a variable called `"list"` to a **List** of the names of Greek letters: `[“alpha”, “beta”, "gamma"]`.



2.15.2 Create List With Item Repeated No. of Times

This action block creates a new **List** with the left-hand input item repeated several times as specified by the number inserted into the right-hand input.

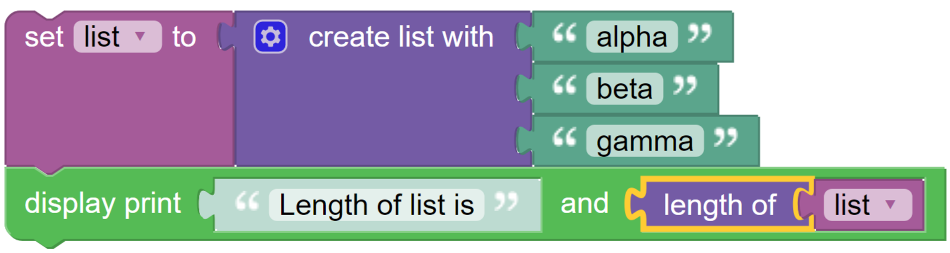


In this example, a variable called ”`list`” is set to a **List** of the number “`123`” repeated 5 time, that is: `[123, 123, 123, 123, 123]`.

2.15.3 Length Of List

This value block calculates the number of items in the input **List**.

In this example the number of items in `"list"` which contains `[“alpha”, “beta”, “gamma”]` is printed on the display as `Length of list is 3`.

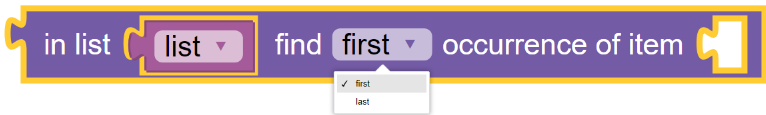


2.15.4 Is Empty

This *Boolean* value block is `True` if the input **List** is empty (i.e. it has no items in it) or is `False` if the **List** has members.

2.15.5 In List Find First / Last Occurrence of Item

This value block searches a **List** for a given item and is set to the index, a numeric integer, in the **List** at which the item was found, if it was found.



A **List** index ranges from `0` to `n-1`, where `n` is the number of items in the **List**. **List** indexing follows the rules of **KookaBlockly**'s underlying Python programming language.

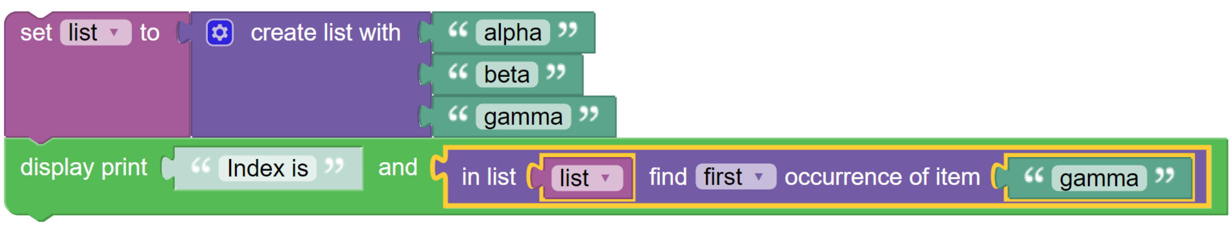
If the item was not found the value block is set to `-1` instead.

The first input socket accepts the variable which is a **List**, and the second input item specifies the value that is being searched for.

The drop-down list gives the choice of finding the `first` or the `last` occurrence of the specified item in the **List**.

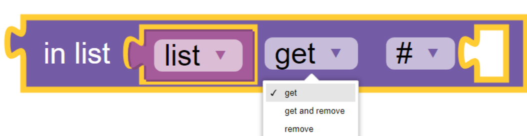
In List Find Example

In this example we search for the first occurrence of `“gamma”` in the **List** `[“alpha”, “beta”, “gamma”]` and print the result on the display as `Index is 2`, `"gamma"` being the third item in the **List**.



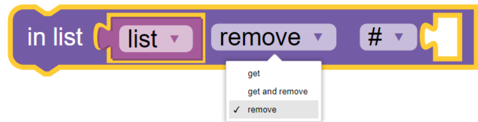
2.15.6 In List Get / Remove Item

This value block operates on a **List** to retrieve, retrieve and remove, or just remove an item at a particular position in the **List**. The value of the **List** item is returned as the result of the block.



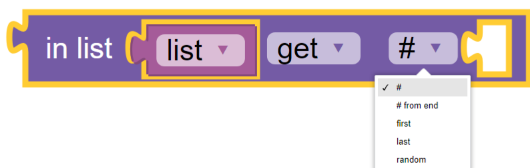
The images show the block and the drop-down list of the operation choices available in the block:

1. **get** fetches the indexed item from the **List** without altering the **List**'s content
2. **get and remove** fetches the indexed item from the **List** and then deletes it from the **List**. The length of the **List** reduces by one.
3. **remove** deletes the indexed item from the **List**. This is an action block and does not return any value.



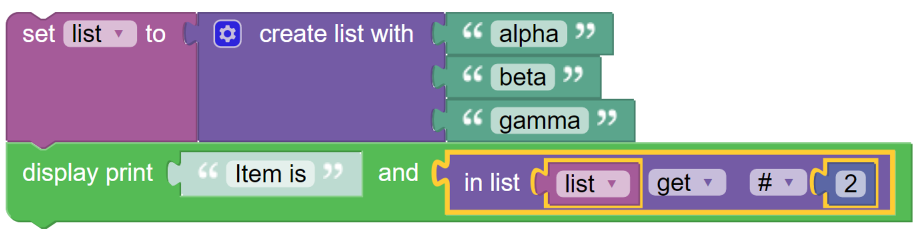
The second drop-down list has a number of choices as to which item in the **List** to get or set:

1. **#** the index of the item in the **List**
2. **# from end** the #th item from the end, where `0` would be the last item, 1 the second-last item etc.
3. **first** the first item in the **List**. The index input will not be present.
4. **last** the last item in the **List**. The index input will not be present.
5. **random** uses a random item from the **List**. The index input will not be present.

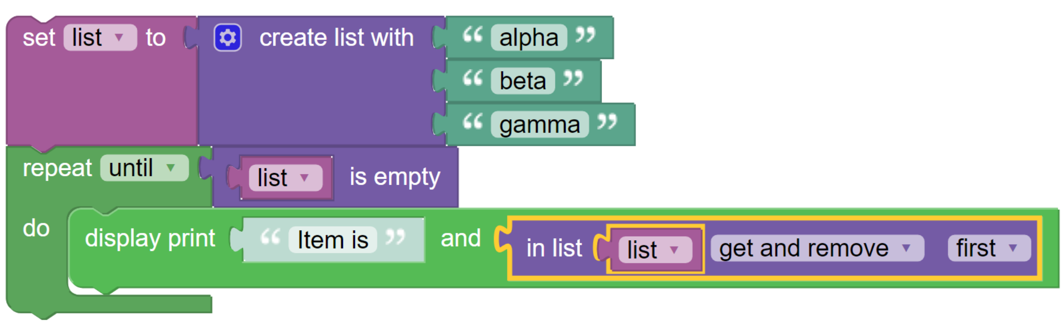


In List Get / Remove Examples

In this example, the variable item is set to the result of getting the item with index 2 from the **List** containing `[“alpha”, “beta”, “gamma”]`. The result is printed on the display as `Item is gamma`.



In this example, items from a **List** containing `[“alpha”, “beta”, “gamma”]`, are removed and printed on the display until the **List** is empty.



2.15.7 In List Set / Insert Item

This action block either changes the value of an item at a specified location to the input value or inserts a new item with the input value at the specified location in a chosen **List**.

The first parameter is a drop-down list with the operation choices:

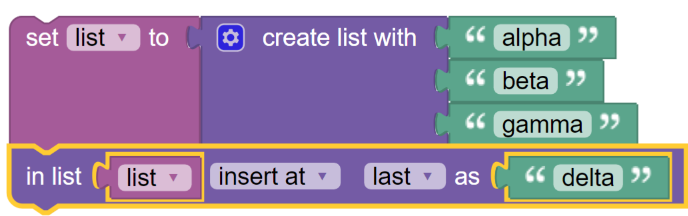
1. **set** writes the input value to the indexed item in the **List**, overwriting its prior value
2. **insert at** creates a new member of the **List** at the indexed position with the input value. The members from the old index onwards are shifted into the next position and the length of the **List** increases by one.

The second drop-down list has a number of choices as to which item in the **List** to set or insert:

1. **#** the index of the item in the **List**
2. **# from end** the #th item from the end, where `0` would be the last item, `1` the second-last item etc.
3. **first** the first item in the **List**. The index input will not be present.
4. **last** the last item in the **List**. The index input will not be present.
5. **random** uses a random item from the **List**. The index input will not be present.

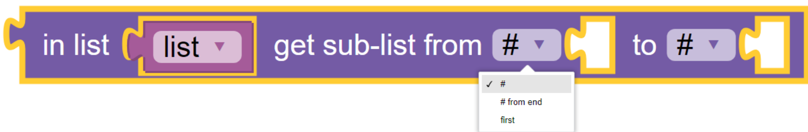
In List Set / Insert Example

By way of example, we may wish to add `"delta"` to the end of the **List** initially containing the values `[“alpha”, “beta”, “gamma”]`.



2.15.8 In List Get Sub-List

This value block copies a portion of a chosen **List** and provides the **Sub-List** as its output.



As for the **Create List** block, a variable is needed to contain the output **Sub-List**.

The **Sub-List** portion starts from the first chosen index and ends at and includes the second chosen index.

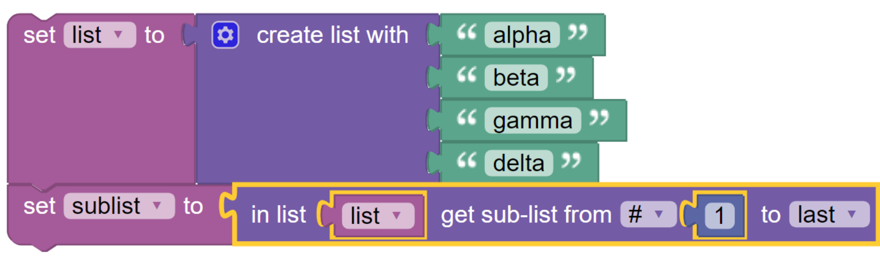
Two drop-down boxes provide options for specifying the beginning index and the ending index:

1. **#** the index of the item in the **List**
2. **# from end** the #th item from the end, where `0` would be the last item, 1 the second-last item etc.
3. **first** the first item in the **List**, only for the beginning index. The index input will not be present.
4. **last** the last item in the **List**, only for the ending index. The index input will not be present.

The beginning index must be less than or equal to the ending index. If not, an error will be raised and the script will terminate.

Get Sub-List Example

In this example a smaller **List** is assigned to variable "sublist" comprising the the items from index number 1 to the last item in the **List** containing `[“alpha”, “beta”, “gamma”, "delta"]`.



The **Sub-List** will contain `[“beta”, ”gamma”, delta”]`.

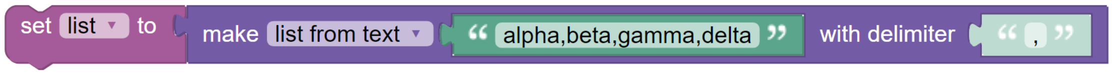
2.15.9 Make List / Text With Delimiter

This value block will, depending on the option chosen in the drop-down list:

1. **list from text** parses a text string into items separated by the delimiter text and arranges the items into a **List**.
2. **text from list** takes the items in a **List** and concatenates them into a text string separated by the delimiter text.

Make List / Text Examples

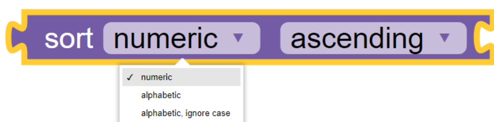
An example is to parse a text string into a **List**. The text string contains the first four Greek letters separated by commas. The results is a **List** of the Greek letters as the variable "letters".



The complementary operation is to generate the original text from the **List** containing `["alpha", "beta", "gamma", "delta"]` and to print it on the **Kookaberry's** display.

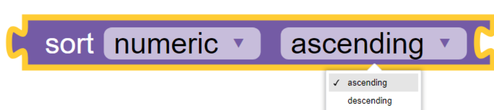
2.15.10 Sort List

This value block allows a **List** to be re-ordered by sorting in numeric or alphabetic order in an ascending or descending format.



The first option is for the type of sorting:

1. **numeric** if the **List** contains numbers, the **List** will be sorted in numeric order
2. **alphabetic** the **List** will be sorted according to the ASCII character codes of the contents. See https://www.ascii-code.com
3. **alphabetic, ignore case** the **List** is sorted into ASCII code order, but all letters are treated as lower-case.

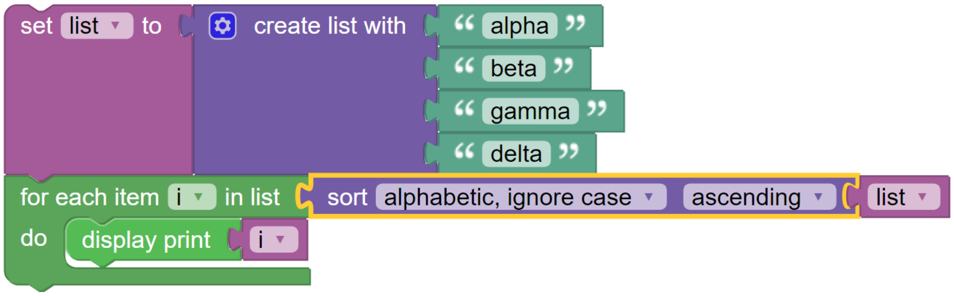


The second option is for the order of sorting:

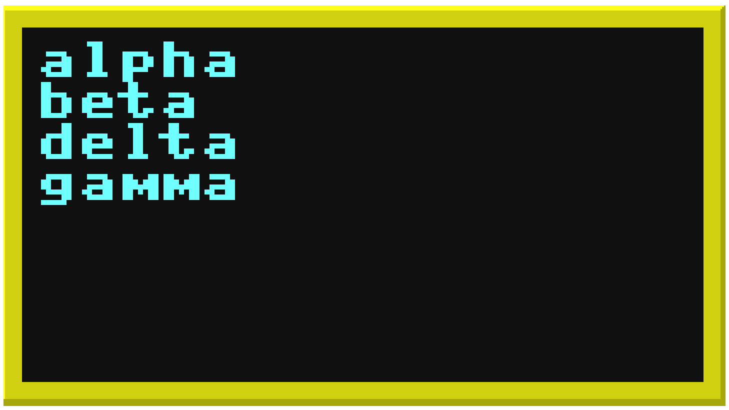
1. **ascending** the **List** is ordered from low to high values
2. **descending** the **List** is ordered from high to low values

Sort List Example

This example prints the items in the **List** containing `[“alpha”, “beta”, “gamma”, "delta"]` on successive rows of the **Kookaberry** display in alphabetical order.



The result of the example can be seen on the **Kookaberry**’s display where the sorted order of the **List** is printed on successive lines:



2.16 Math

Fundamental to any computer program is the ability to do mathematical computations.

The **Math** Category provides the repertoire of mathematical functions shown in Fig. 2.31.

2.16.1 Number

This value block represents a fixed number that is specified by editing the default number `123` in the block.

The number can be any valid integer or floating point number:

- the number can be signed, that is, preceded by the character + (default and assumed if not present) or the character `-` for negative numbers
- there is no limit (other than computer memory) for how large the number can be
- an integer in the form `123456`

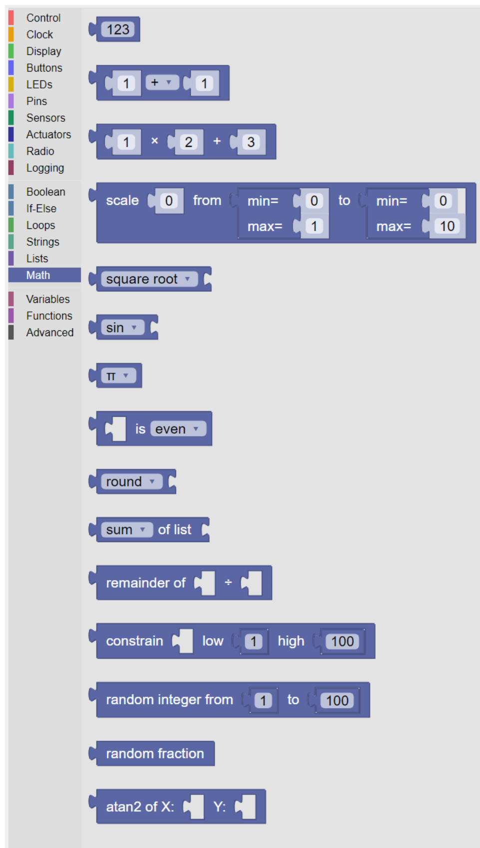


Fig. 2.31: The palette of **KookaBlockly Math** blocks

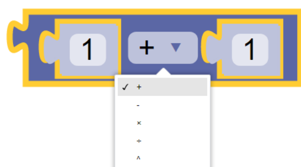
- a floating point number in the form `123456.789`
- scientific notation in the form `1.234567e5` can be used and will be displayed in integer or floating point form as appropriate `123456.7`

Number Example

This example prints a number on the **Kookaberry**'s display:

2.16.2 Arithmetic

This value block operates on two input values or value blocks that represent numbers with the chosen arithmetic operator.

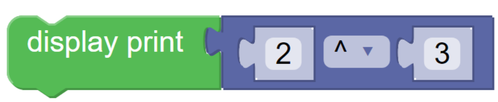


The operations that can be chosen from the drop-down list are:

1. addition (**+**)
2. subtraction (**-**)
3. multiplication (**x**)
4. division (**÷**)
5. and raised to the power of (**^**)

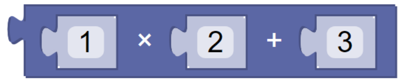
Arithmetic Example

This example prints the result of `2` raised to the power of `3` (ie. `2` cubed which is `8`) on the **Kookaberry**'s display:

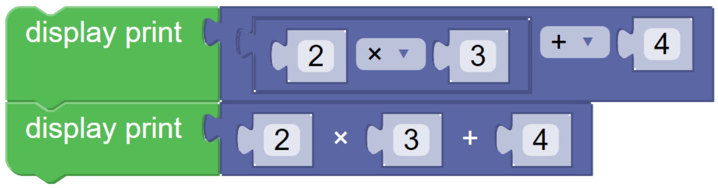


2.16.3 Multiply and Add

This value block multiplies the first numerical value block input by the second numerical value block input and then adds the third numerical value input to the product of the first two inputs.

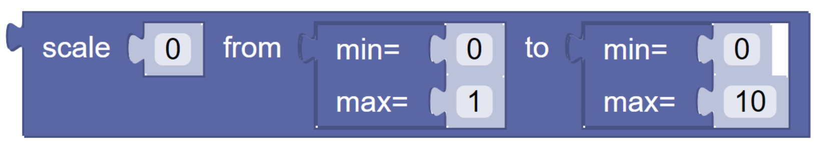


This block is a convenient way to achieve the same result as using two **Arithmetic** blocks as in the example below. Both blocks will print the same result (`10`).



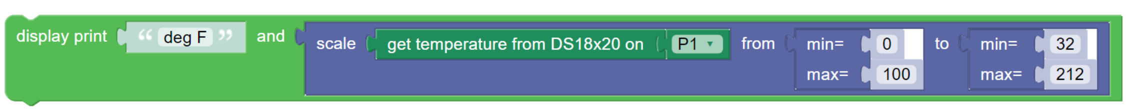
2.16.4 Scale Function

The **Scale** value block will perform the necessary computations to convert the number on the first input from a scale defined by the second input, to another scale defined by the third input.



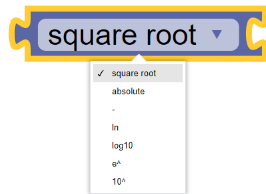
Scale Example

By way of example, this script using the **Scale** block will convert a Celsius water temperature sensor reading (range freezing point `0` to boiling point `100`) into the equivalent degrees Fahrenheit (range freezing point `32` F to boiling point `212` F) and print it on the **Kookaberry**'s display.



2.16.5 Math Function

This value block performs the chosen mathematical function on the numerical value input.



The options that are available are:

1. **square root** - gives the number that when multiplied by itself is equal to the input - see https://en.wikipedia.org/wiki/Square_root
2. **absolute** - the unsigned magnitude of the input value - see https://en.wikipedia.org/wiki/Absolute_value
3. **-** - changes the input number's sign from positive to negative or negative to positive - the same as multiplying by `-1`
4. **ln** - natural (base e) logarithm of the input number - see https://en.wikipedia.org/wiki/Natural_logarithm
5. **log10** - base 10 logarithm of the input number - see https://en.wikipedia.org/wiki/Logarithm
6. **e^** - the constant e raised to the power of the input number - see https://en.wikipedia.org/wiki/Exponential_function
7. **10^** - 10 raised to the power of the input number - see https://en.wikipedia.org/wiki/Exponentiation

2.16.6 Trigonometric Function

This value block performs the basic selected trigonometric functions. on the input numerical angles.

The functions available for selection in the drop-down list are:

1. **sin** - sine of the input angle - see https://en.wikipedia.org/wiki/Sine_and_cosine
2. **cos** - cosine of the input angle - see https://en.wikipedia.org/wiki/Sine_and_cosine
3. **tan** - tangent of the input angle - see https://en.wikipedia.org/wiki/Trigonometric_functions
4. **asin** - arc-sine of the input value - the inverse of sine.
5. **acos** - arc-cosine of the input value - the inverse of cosine.
6. arc-tangent (`atan`) of the input value - the inverse of tangent.

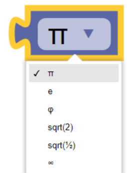
The functions **sin**, **cos** and **tan** expect the input to be in degrees. The outputs for these functions are floating point numbers between `-1` and `+1` inclusive.

The inverse functions **asin**, **acos** and **atan** expect the input to be floating point numbers between `-1` and `+1`. The outputs will be in degrees ranging from `-180` to `+180` inclusive.

See also https://en.wikipedia.org/wiki/Trigonometric_functions

2.16.7 Special Constants

This value block provides several special constants that are important and often used numbers in mathematics.



To choose a constant use the drop-down list and select from

1. - pi used in dealing with circles - see https://en.wikipedia.org/wiki/Pi
2. **e** - Euler's number used in exponential function - see https://en.wikipedia.org/wiki/E_(mathematical_constant)
3. - the Golden Ratio phi - see https://en.wikipedia.org/wiki/Golden_ratio
4. **sqrt(2)** - the square root of 2 - see https://en.wikipedia.org/wiki/Square_root_of_2
5. **sqrt(½)** - the square root of ½ - see https://en.wikipedia.org/wiki/Square_root_of_2#Multiplicative_inverse
6. ∞ - infinity - see https://en.wikipedia.org/wiki/Infinity

For a list of most of the mathematical special constants see https://en.wikipedia.org/wiki/List_of_mathematical_constants

2.16.8 Number Property Test

This value block gives a *Boolean* value of `True` or `False` depending on whether the numerical input value has the chosen property or not.

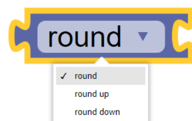
The property to test is selected from the drop-down list which includes:

1. **even** - whether the input is divisible by `2` - see https://en.wikipedia.org/wiki/Parity_(mathematics)
2. **odd** - whether the input is not divisible by `2` - see https://en.wikipedia.org/wiki/Parity_(mathematics)
3. **prime** - whether the input is divisible only by `1` and itself - see https://en.wikipedia.org/wiki/Prime_number
4. **whole** - whether the input when divided by `1` leaves no remainder - see https://en.wikipedia.org/wiki/Whole_number
5. **positive** - whether the input is greater than `0` - see https://en.wikipedia.org/wiki/Sign_(mathematics)
6. **negative** - whether the input is less than `0` - see https://en.wikipedia.org/wiki/Sign_(mathematics)

7. **divisible by** - whether the input when divided by the number in the second input leaves no remainder. If **divisible by** is selected it will add a second input socket for the number to test against. - See https://en.wikipedia.org/wiki/Remainder

2.16.9 Round Number

This value block rounds the numerical input value to a whole number using the chosen method.



The method is chosen from the block's drop-down list:

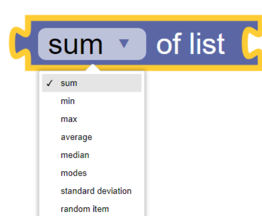
1. **round** - rounds the number in the standard manner, if the fraction is greater than or equal to 0.5 it rounds up to the next more positive whole number, and if the fraction is below 0.5 the block rounds down towards the negative direction.
2. **round up** - if there is a fractional component the block always rounds up to the next more positive whole number.
3. **round down** - removes any fractional component.

Input numbers are floating point and output numbers are integers.

- **round up** means in the positive direction.
- **round down** means in the negative direction.

2.16.10 List Operations

This block computes a mathematical function based on the content of a **List** which is connected to the input to the block.



The function to be used is selected from the drop-down list:

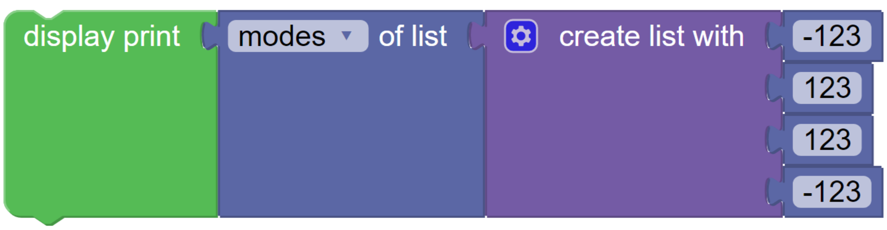
1. **sum** - computes the arithmetic sum of the members of the **List** - see https://en.wikipedia.org/wiki/Summation
2. **minimum** - returns the number with the minimum value from the **List** - see https://en.wikipedia.org/wiki/Maximum_and_minimum
3. **maximum** - returns the number with the maximum value from the **List** - see https://en.wikipedia.org/wiki/Maximum_and_minimum

4. **average** - returns the arithmetic mean of the items in the **List** - see https://en.wikipedia.org/wiki/Arithmetic_mean
5. **median** - returns the arithmetic median of the items in the **List** - see https://en.wikipedia.org/wiki/Median
6. **modes** - returns a **List** of the most numerous items in the **List** (example below) - see https://en.wikipedia.org/wiki/Mode_(statistics)
7. **standard deviation** - computes the statistical standard deviation of the items in the **List** - see https://en.wikipedia.org/wiki/Standard_deviation
8. **random item** - returns an item from the **List** that has been selected at random - see also https://en.wikipedia.org/wiki/Random_variable

Note: All functions except **modes** and **random** require that the input **List** contain only numerical or *Boolean* items. **Boolean** items are evaluated as `False` = `0` and `True` = `1`. The **modes** and **random** functions accept **Lists** with members of any type, i.e. numeric integer and floating point, boolean, and character strings.

List Operations Example

This is an example of the use of **modes**. The input **List** contains `[-123, 123, 123, -123]`. The block returns a **List** of the most numerous items in the **List**, being `[-123, 123]`. If we changed the input **List** to `[-123, -123, 123, -123]`, the block would return `[-123]`, a **List** of one item being the most numerous.



2.16.11 Remainder

This block returns the fractional portion of the number that results when the number at the first input is divided by the number at the second input.

For example, when `3` is divided by `2` the result is `1.5`. The remainder is the fractional portion which is `0.5`.

See also https://en.wikipedia.org/wiki/Remainder

2.16.12 Constrain

This block constrains the number at the first input to be between the minimum number defined as the second input and the maximum number defined as the third input.

There are three possible outputs from this block:

1. if the input number is less than the minimum number, the output will be set to the minimum number.
2. if the input is between the minimum and maximum inclusive, the number is passed through as-is.
3. if the input number is greater than the maximum number, the output will be set to the maximum number.

2.16.13 Random Integer

This block generates an integer number that is constrained to be from a minimum integer defined by the first input, and a maximum integer defined by the second input.

For example, to simulate the roll of a six-sided die, set the minimum to `1` and the maximum to `6`.

See also https://en.wikipedia.org/wiki/Random_variable

2.16.14 Random Fraction

This value block creates a random floating point number from 0 up to but not including 1.

See also https://en.wikipedia.org/wiki/Random_variable

2.16.15 Atan2 of X

This value block returns the arc tangent of two numerical values at inputs x and y .

This function is similar to calculating the arc tangent of y/x, except that the signs of both arguments are used to determine the quadrant of the result. The result is an angle expressed in degrees in the range `-180` to `+180`.

See also https://en.wikipedia.org/wiki/Atan2

2.17 Variables

Variables are a way of creating and manipulating a named value, in the same way that algebra uses names to refer to a value. A **Variable** is useful as a named container to store a value for later use in one or more places in a **KookaBlockly** script.

Examples of typical **Variable** names are **X**, **Y** and **Z** when referring to cartesian coordinates; **H**, **W** and **D** as dimensions of an object; and **i** or **j** as an index into a **List**. **Variable** names can of course be longer, for example **height**, or **temperature**

When **KookaBlockly** is first started, or when a new script is started, the **Variables** palette looks like this Fig. 2.32.

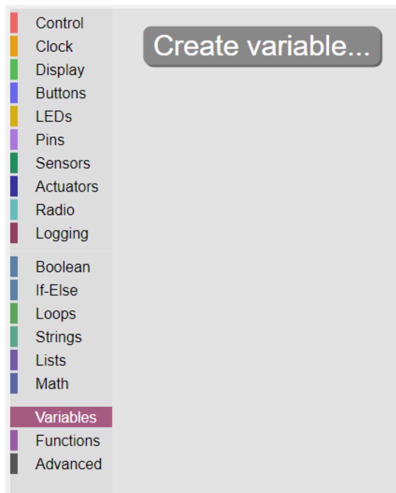


Fig. 2.32: The initial **Variables** palette

2.17.1 Create Variable

Clicking on "Create variable" brings up a dialogue box, shown in Fig. 2.33, where the user can define the **Variable**'s name. Type in a name and then click on **OK**. The figure shows an example name `"my_variable"`.

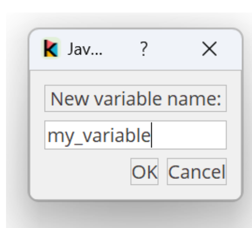
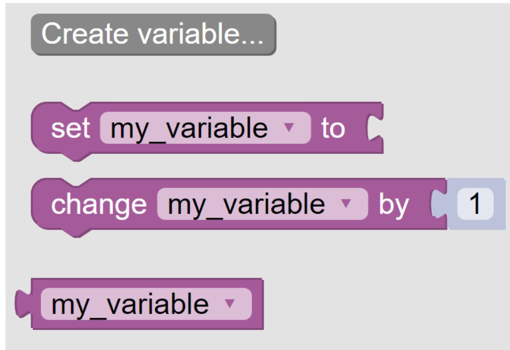
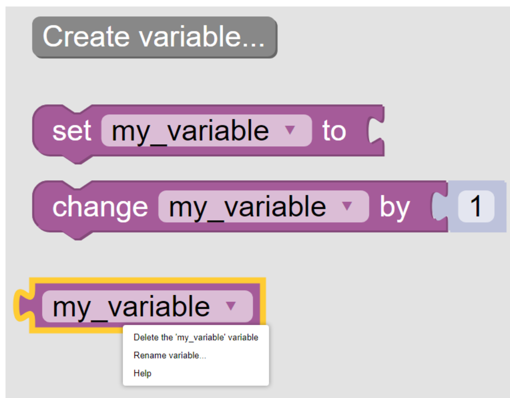


Fig. 2.33: Creating a **Variable** named my_variable

Once a new **Variable** has been created, the new **Variable** will be available in the **Variables** palette.



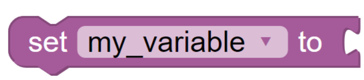
It is possible to right-click while hovering over the **Variable** block in the palette to reveal a number of actions which can be selected by then clicking on them:



1. **Delete the variable** - removes the **Variable**, and its associated blocks if it was the only **Variable**.
2. **Rename the variable** - brings up a dialogue box, as for creating a **Variable**, in which the new name can be typed. The new name must contain at least one visible character and not be the same as any other **Variable**.
3. **Help** - this option does not yet work. It is intended eventually to display Help text.

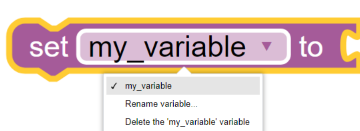
2.17.2 Set Variable

Using this block, a value can be assigned to a **Variable** by attaching a value block to its input. The value can be a number, a boolean, or a character string.



The **Variable** to be assigned the value can be selected from the drop-down-list.

The drop-down list also has some other choices:

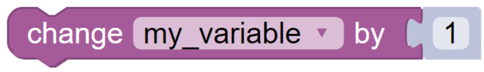


1. **Rename variable** - brings up a dialogue box in which the new name can be typed. The new name must comprise at least one visible character and must not be a duplicate name.

2. **Delete the variable** - removes the **Variable** and its associated blocks from the script.

2.17.3 Change Variable

This action block allows the user to change the selected **Variable** by a number specified by the input numerical value.



This block will only work for numerical variables and will only accept numerical values.

Character strings and boolean values will not be accepted.

The example in Fig. 2.34 illustrates how this block may be used as a counter.

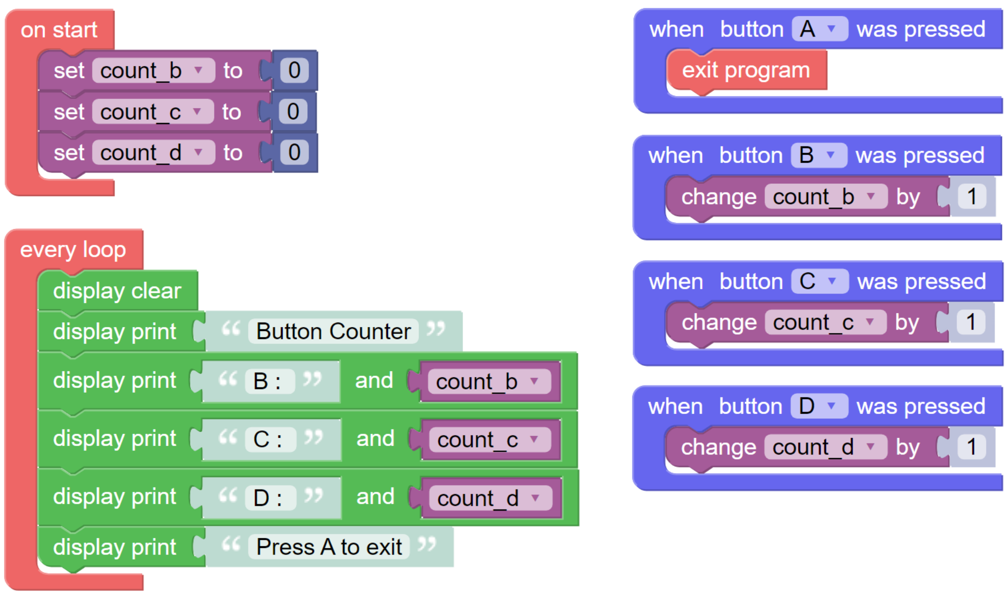


Fig. 2.34: Example script counts button presses

Three variables are set up: `count_b`, `count_c` and `count_d` to count the number of times buttons B, C and D are pressed.

The running totals are printed on the **Kookaberry**'s display, as shown in Fig. 2.35.

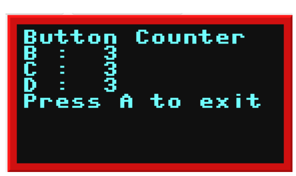
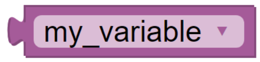


Fig. 2.35: The Kookaberry display resulting from Fig. 2.34

2.17.4 Variable Value

This value block allows a user to attach a variable's value to the input of another block.



The example in Fig. 2.36 reads a temperature from a sensor once per 5 seconds, storing it in a **Variable** named `"temperature"`, then using the stored value to perform a conversion calculation and display the original and converted values on the **Kookaberry** display:

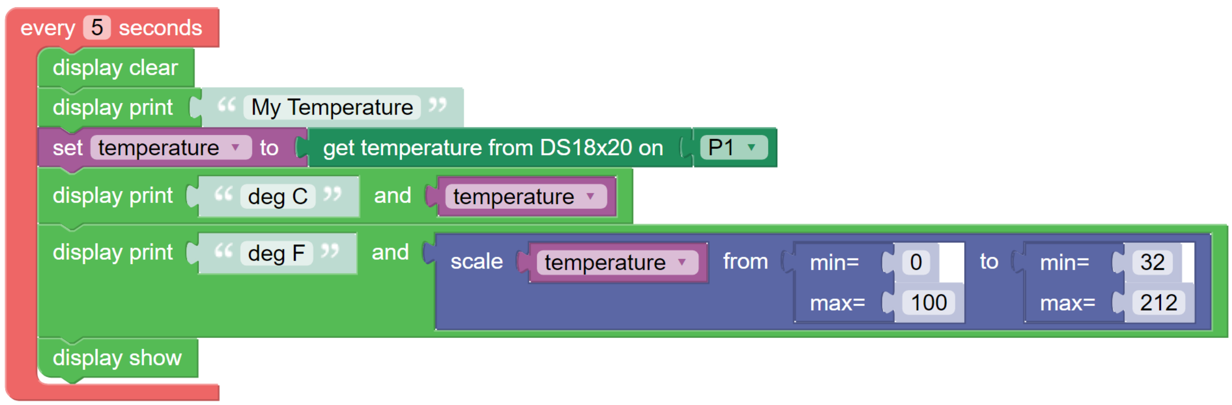


Fig. 2.36: Example script reads converts temperature readings to Fahrenheit

2.18 Functions

Functions are blocks that contain a sequence of other blocks.

Once defined, functions are available on the **Functions** palette for use in the **KookaBlockly** script in which they are defined. See Fig. 2.37.

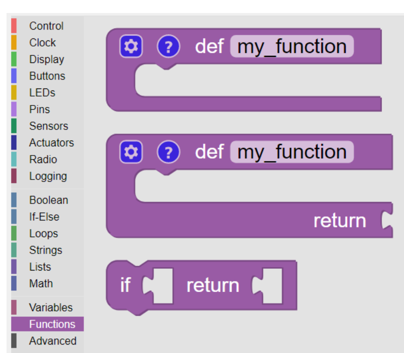


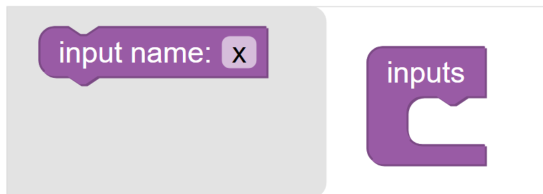
Fig. 2.37: The initial **Functions** palette

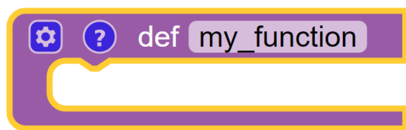
Function blocks can be used repeatedly in a script without needing to repeat all the blocks they contain. This simplifies scripts and saves valuable computer memory space.

Important: The function definition must remain in the **KookaBlockly** workspace for it to remain available in the **Functions** palette. Deleting the function definition will remove the function block from the palette and all instances of it from the script.

2.18.1 Define Function

This block allows a user to define a sequence of blocks that will be run together when the function's block is used.

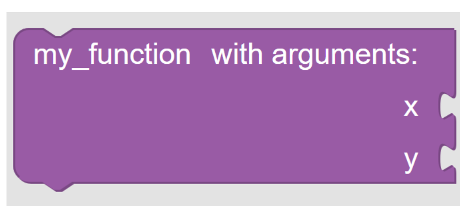




To define a function, drag this block into the **KookaBlockly** workspace.

The block has a gear wheel which when clicked causes the definition box to appear:

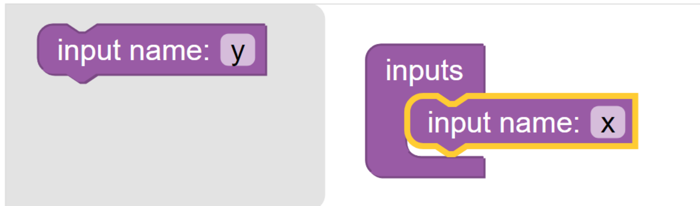
Once the definition of the function block is complete, click on the cog symbol once again to close the definition box. Remember to leave the function definition block in the **KookaBlockly** workspace!

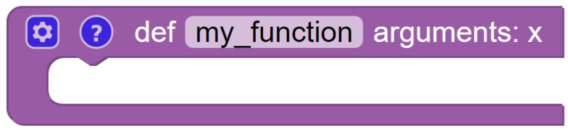


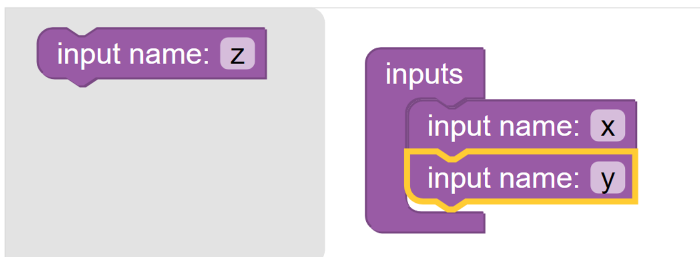
The function block will then be available in the palette for use elsewhere in the script:

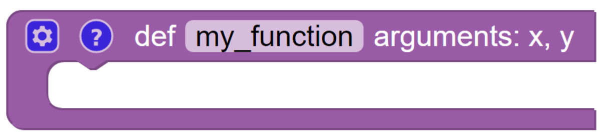
Define Inputs

A function may, or may not, have inputs that will be used by the script inside the function.









To define the inputs, drag the input block on the left of the box into the bracket on the right.

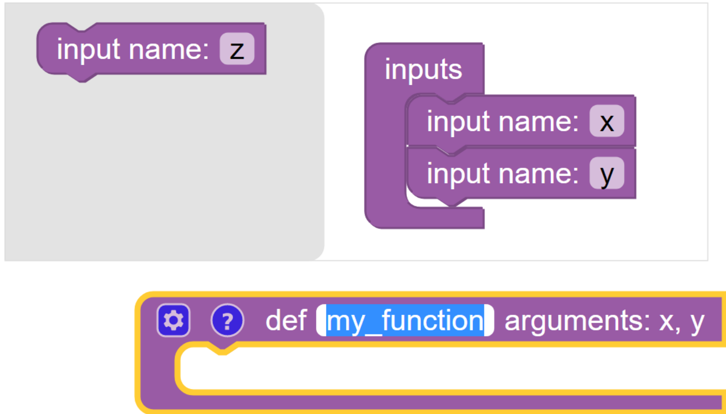
To remove an input, drag the input block out of the bracket back to the grey box on the left.

Rename the inputs as desired by editing their names (click on the name and type the new name). It is best to give the inputs names that are meaningful so the **KookaBlockly** script can be more easily understood by humans.

All the inputs will become *Variables*, do take care not to duplicate their names!

Function Name

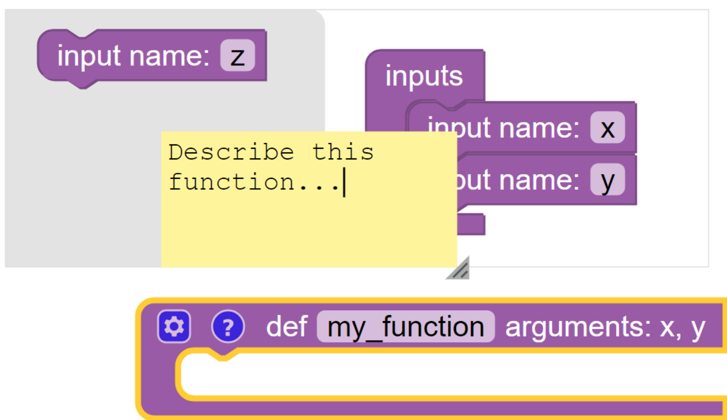
Functions must have unique names within the context of the **KookaBlockly** script they are in.



To define the function name, click on its name and edit the text.

Function Description

Functions can optionally be described. A description may say what the function does, what its inputs are, what computations it performs, and what its output is.



Click on the question mark, **?**, and a description box will appear. Type the description in the box.

To close the description box, click on the question mark.

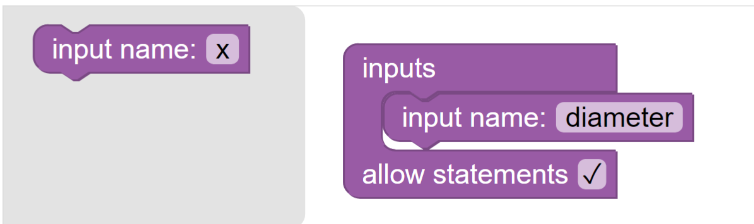
To view the description, click on the question mark and click again to close the description.

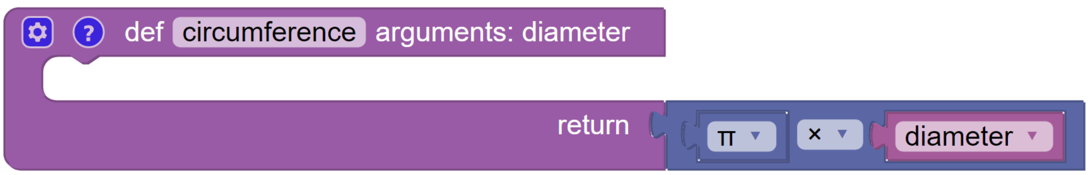
2.18.2 Define Function with Return Value

This block works in a similar manner to the **Define Function** block except that this block returns a value.

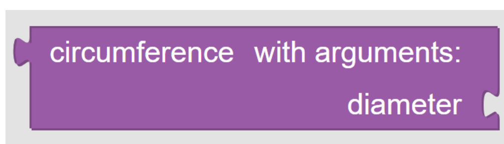
The value returned is the output of the value block socketed at the bottom of the **Define Function with Return Value** block.

Here is an example where a function is defined to calculate the circumference of a circle given a radius:





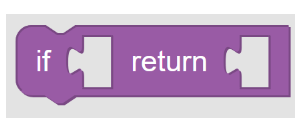
Once the definition of the function block is complete, click on the cog symbol once again to close the definition box. Remember to leave the function definition block in the **KookaBlockly** workspace!



The function block will then be available in the palette for use elsewhere in the script:

2.18.3 If Condition Return

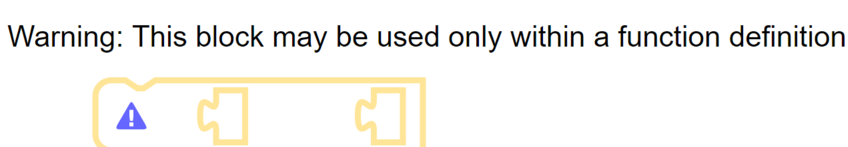
This block can be used in both the **Function Definition** and **Function Definition With Return Value** blocks.



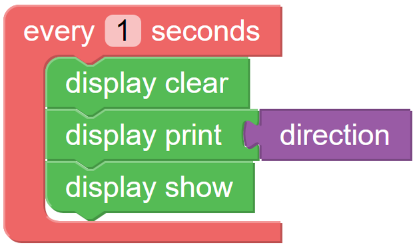
It will check the True / False condition in the first value block input and if it is `True` it will end the function immediately, returning the value in the second input .

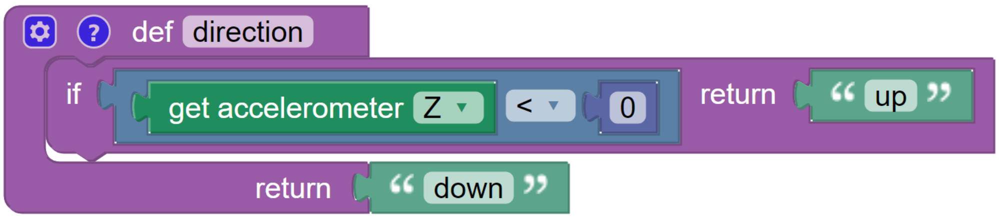
If used inside a Function Definition block (without a return value) the returned value input will not be available. Instead the block will just end the function if the input condition is `True`.

This block cannot be used outside of the **Function Definition** blocks. If this is attempted the block will be blanked out.



The following is an example of the use of the **If Condition Return** block with a function named `direction`.





The function tests the sign of the acceleration read from the Z axis of the internal accelerometer. If Z acceleration is negative then the tested condition is `True` which means the **Kookaberry** is facing up, and the string `"up"` is returned. Otherwise, that is the condition is `False`, which means the **Kookaberry** is tilted face-down. The function completes and returns the string `"down"`.

The main script is a loop which repeats every second and prints the value of the function on the display. The display will change as the **Kookaberry** is oriented face-up or face-down.

2.19 Advanced

The Advanced Category is provided to extend the capability of **KookaBlockly** by allowing the definition of additional blocks using Python programming statements. See Fig. 2.38.

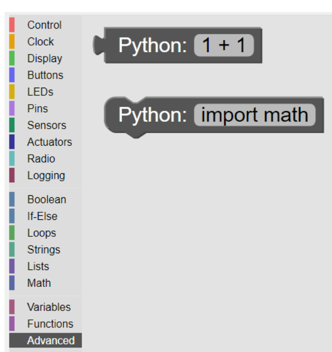


Fig. 2.38: The Advanced block palette

This category is available to the more advanced user as a way of transitioning from **KookaBlockly** to Python scripts, and also to add extended functionality such as using special sensors and actuators and other **Kookaberry** peripherals, or using Python module libraries.

Important: When typing in the Python statement, please do not use the single quotation mark `'` as this will cause the saved script to not be loaded back in from file correctly. Always use the double quotes `"` character, as in the example shown at the end of this section.

2.19.1 Python Value

This value block allows the result of any Python statement to be passed to **KookaBlockly** block input sockets.

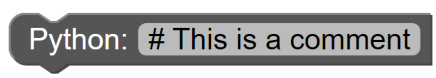
The Python statement is typed into the text box in the block. In the default block, the statement `1+1` results in the output value of `2`.

2.19.2 Python Action

This action block permits any Python statement to be inserted into a **KookaBlockly** script. The statement is typed into the text box in the block.

Typical usage might be to import a library module, for example "`import math`", or "`import mymodule`" where a customised module has been developed, or anything else that is permitted in Python syntax.

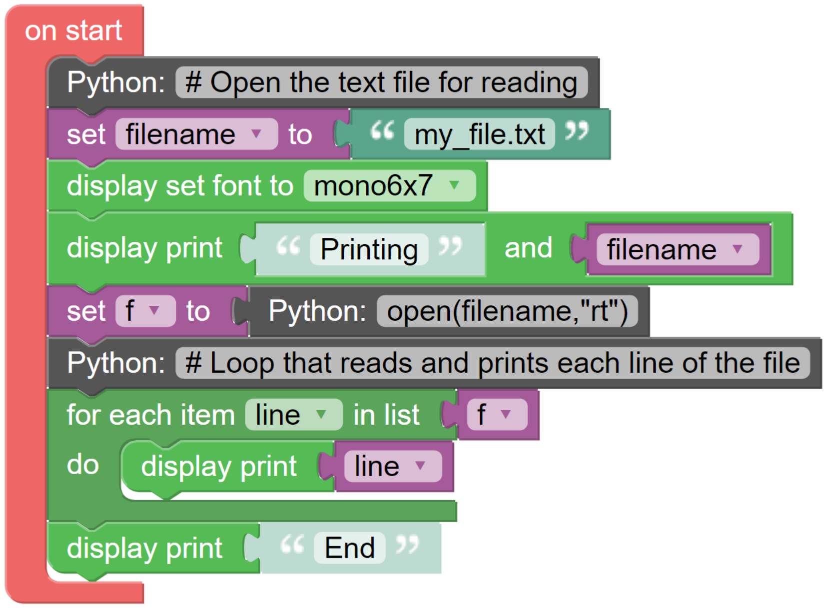
It can also be used to insert comments into the script by prefixing the inserted text with a # character, designating that the following text is a comment.



2.19.3 Advanced Example

KookaBlockly does not, at this stage, provide any blocks to read a text file.

This example reads a plain text file using the **Advanced** blocks and prints each line that is read on the display.



This script uses two **Python Action** blocks to insert in-line comments in the **KookaBlockly** and the resulting MicroPython script.

Three variables need to be created:

1. `filename` which is set to a string containing the files' name `"my_file.txt"`
2. `f` which is used to store a **List** of lines coming from the text file
3. `line` which temporarily stores each line from the file as they are read in the loop.

Only one **Python Value** block is needed that sets the variable `f` to a **List** of lines created by opening the text file using a Python statement.

The MicroPython code that the **KookaBlockly** script generates is shown below.

```
import machine, kooka
import fonts

filename = None
f = None
line = None

# On-start code, run once at start-up.
if True:
  # Open the text file for reading
  filename = 'my_file.txt'
  kooka.display.setfont(fonts.mono6x7)
  kooka.display.print('Printing', filename, show=0)
  f = open(filename,'rt')
   # Loop that reads and prints each line of the file
```

(continues on next page)

(continued from previous page)

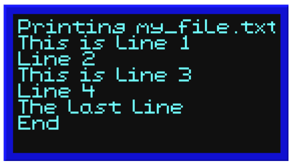
```
  for line in f:
    kooka.display.print(line, show=0)
  kooka.display.print('End', show=0)

# Main loop code, run continuously.
while True:
  kooka.display.show()
  machine.idle()
```

To run the above script, a text file called `my_file.txt` should be stored on the **Kookaberry**'s file system in its root folder. The file contains the following:

```
This is line 1
Line 2
This is line 3
Line 4
The last line
```

When the script is run, the appearance of the **Kookaberry** display is as below:



GLOSSARY OF TERMS

This glossary contains the definitions of terms used throughout this KookaBlockly Reference Guide and is intended to demystify the vocabulary often used in association with computers and software.

Kookaberry
The **Kookaberry** is a microcomputer specifically designed for **STEM** educational applications. See https://learn.auststem.com.au/exploring-the-kookaberry/

KookaSuite
A suite of programming tools for the **Kookaberry** comprising **KookaBlockly** visual coding tool, **KookaIDE** a **MicroPython** integrated development tool, and **KookaTW** a tool for mirroring / virtualising the **Kookaberry**'s display and buttons.

Visual Code Editor
A visual code editor allows users to work with code visually but still involves actual code blocks or snippets. It might use drag-and-drop interfaces, code blocks, or other visual elements to assist in code creation. Visual code editors often aim to make coding more accessible to beginners or those who are not familiar with traditional text-based coding environments. It differs from a graphical code editor that may involve more abstract graphical representations of code structures, while visual code editors usually retain a connection to the actual code, using visual elements to enhance the coding experience. See also https://en.wikipedia.org/wiki/Visual_programming_language

OLED
Organic Light Emitting Diode - the lighting technology that is used in the **Kookaberry**'s display - see https://en.wikipedia.org/wiki/OLED

LED
Light Emitting Diode - a semiconductor that emits a specific wavelength of light when energised. The **Kookaberry** has three LEDs on the front under the display. They emit red, yellow and green light. There are two further LEDs on the back: a green LED indicating the **Kookaberry** has power, and a blue LED which indicates file writing activity, or if pulsing slowly indicates the **Kookaberry**'s power supply voltage is low. See also: https://en.wikipedia.org/wiki/Light-emitting_diode

GPIO
General Purpose Input and Output - the electrical signals to and from a microcomputer are connected by these, and are referred to as *Pins* by **KookaBlockly**. See also https://en.wikipedia.org/wiki/General-purpose_input/output

USB
Universal Serial Bus - a communications and power connection used by the **Kookaberry** to communicate with the programming personal computer, and the receive power. See also https://en.wikipedia.org/wiki/USB.

MicroPython
A variant of the computer programming language **Python** developed for use on micro-computers. The **Kookaberry** is programmed using **MicroPython** and has a built-in compiler accessible through editors such as

KookaIDE and **Thonny**. **KookaBlockly** automatically generates **MicroPython** code when the user assembles a script from **KookaBlockly**'s visual blocks. See also https://en.wikipedia.org/wiki/MicroPython

Python
A high-level computer programming language that was designed to be easy to use and easily comprehended. It nonetheless is a very powerful language and is now favoured by educational institutions as the first-taught computer language. See also https://en.wikipedia.org/wiki/Python_(programming_language)

IDE
Integrated Development Environment - a software application that integrates code editing, testing and sometimes code debugging tools. Examples relevant to **KookaBlockly** and the **Kookaberry** are **KookaIDE** and **Thonny**. See also https://en.wikipedia.org/wiki/Integrated_development_environment

STEM
Science, Technology, Engineering and Mathematics - an umbrella term to group these disciplines in the context of education and career development. See also https://en.wikipedia.org/wiki/Science,_technology,_engineering,_and_mathematics

Raspberry Pi Pico
A microcomputer developed by the **Raspberry Pi Foundation** based on their **RP2040** microprocessor chip. The **RP2040** microprocessor chip is used in later hardware versions of the **Kookaberry**. See also https://en.wikipedia.org/wiki/Raspberry_Pi

STM
STMicroelectronics N.V. commonly referred to as ST or STMicro is a multinational corporation and technology company of French-Italian origin. **STM** microprocessors are used in the original hardware version of the **Kookaberry**. See https://en.wikipedia.org/wiki/STMicroelectronics and https://en.wikipedia.org/wiki/STM32

Micro:Bit
A microcomputer for **STEM** applications developed in the United Kingdom by the BBC (British Broadcasting Corporation). It also is programmed using **MicroPython**, and has two official visual programming tools, being **Microsoft MakeCode**, and **Scratch**. The **Micro:Bit** differs from the **Kookaberry** in that it can contain only one program at a time, it has just two buttons and an 8x8 LED matrix display, and it has no electrical sockets with which to connect peripherals, relying instead on using alligator clips or an expansion board. See also https://en.wikipedia.org/wiki/Micro_Bit and https://en.wikipedia.org/wiki/Scratch_(programming_language)

Windows
A personal computer operating system licensed by **Microsoft**. **KookaSuite** will run on **Windows** V10 and later versions. See https://en.wikipedia.org/wiki/Microsoft_Windows

MacOS
A personal computer operating system developed by **Apple**. **KookaSuite** will run on MacOS V13 and later versions using the **Intel** and **Apple**'s M processors. See also https://en.wikipedia.org/wiki/MacOS

Raspbian
Latterly named Raspberry Pi OS, a personal computer operating systems for the Raspberry Pi microcomputer licensed by the **Raspberry Pi Foundation**. **Raspbian** is based on the **Debian Linux** operating system. See also https://en.wikipedia.org/wiki/Raspberry_Pi_OS

Thonny
An open-source Integrated Development Environment tool tailored for programming in **Python**. See https://en.wikipedia.org/wiki/Thonny

Firmware
Low-level computer software that is stored on on-board non-volatile memory. It performs basic low-level tasks to control and monitor the computer hardware, and to make it accessible to high-level software, such as **MicroPython**. **Firmware** updates may sometimes be issued that extend the functionality of a computer, or to remedy bugs or security weaknesses in the **firmware**. The **Kookaberry**'s **firmware** is updated from time to time for the same reasons. See also https://en.wikipedia.org/wiki/Firmware

Real Time Clock (RTC)
A specialised clock chip that keeps precise time. **RTCs** can be built into a microcomputer and / or be connected externally. Often external **RTCs** have a small battery that keeps the clock running when the microcomputer is turned off. The microcomputer can then synchronise its internal **RTC** with the battery-powered external **RTC**. See also https://en.wikipedia.org/wiki/Real-time_clock

ASCII
American Standard Code for Information Interchange - a character encoding standard for electronic communication. **ASCII** codes represent text in computers, telecommunications equipment, and other devices. **MicroPython** uses **ASCII** code when encoding character strings. See also https://en.wikipedia.org/wiki/ASCII

CSV
Comma-Separated-Values - a text file format in which each line contains alphanumeric text data which are separated by commas. The first line of the files can be used to represent headings for the data item columns that are in the following lines. **CSV** formatted files are recognised and can be directly opened by spreadsheet programs. See also https://en.wikipedia.org/wiki/Comma-separated_values

GitHub
A software platform that allows developers to create, store, and manage their code. **GitHub** was acquired by **Microsoft** in 2018. It is commonly used to host **open-source software** development projects. **KookaSuite** and the **Kookaberry firmware** are both distributed using **GitHub**. This document is also maintained and distributed using **GitHub** and **Read the Docs**. See also https://en.wikipedia.org/wiki/GitHub

Read the Docs
Read the Docs is an open-source free software documentation repository and hosting platform. This document is hosted on Read the Docs. See also https://en.wikipedia.org/wiki/Read_the_Docs

Open-Source
Open source is **software** source code, **hardware** designs, documentation, artworks or other intellectual products that are made freely available for possible modification and redistribution, under certain licensing conditions, in a spirit of sharing and collaboration for the greater good. See also https://en.wikipedia.org/wiki/Open_source

Software and Hardware
Software is a collection of programs and data that tell a computer how to perform specific tasks. **Software** often includes associated **software documentation**. This is in contrast to **hardware**, which comprises the physical components from which the system is built and which actually performs the computing work. See also https://en.wikipedia.org/wiki/Software and https://en.wikipedia.org/wiki/Computer_hardware

Example Scripts
All the scripts used in this guide are available for downloading from Github and following the instructions on the README page:

Errata
If errors or issues are found in the **KookaBlockly Reference Guide** please post an issue on GitHub.

Copyright
Blockly is a library from **Google** for building beginner-friendly block-based programming languages.

Kookaberry and **Kooka** are trademarks of Kookaberry Pty Ltd, Australia.

The **Kooka Firmware** release v1.10.0 and **KookaSuite** were created by Damien George (George Electronics Pty Ltd – **MicroPython**) in collaboration with Kookaberry Pty Ltd and the AustSTEM Foundation Ltd.

INDEX

A

ASCII, 115

C

CSV, 115

F

Firmware, 114

G

GitHub, 115
GPIO, 113

I

IDE, 114

K

Kookaberry, 113
KookaSuite, 113

L

LED, 113

M

MacOS, 114
Micro:Bit, 114
MicroPython, 113

O

OLED, 113
Open-Source, 115

P

Python, 114

R

Raspberry Pi Pico, 114
Raspbian, 114
Read the Docs, 115
Real Time Clock (*RTC*), 115

S

Software and Hardware, 115
STEM, 114
STM, 114

T

Thonny, 114

U

USB, 113

V

Visual Code Editor, 113

W

Windows, 114