DCS
Power To Design The Desired

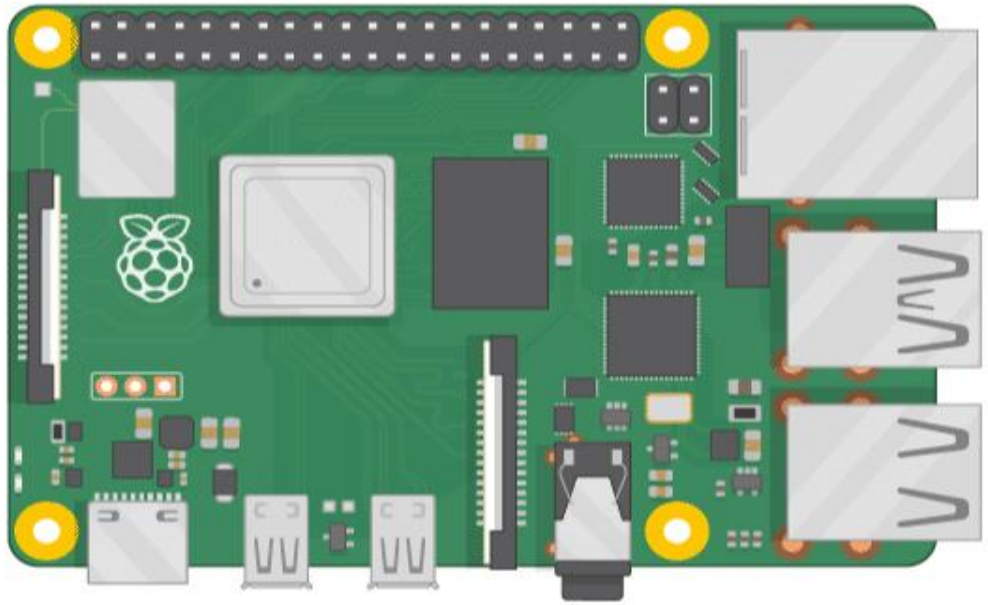
CENTRE FOR SKILL AND ENTREPRENEURSHIP DEVELOPMENT

IIOT3 – RASPBERRY PI

TECHNOLOGY PARTNER

DASSAULT SYSTEMES ptc Mastercam

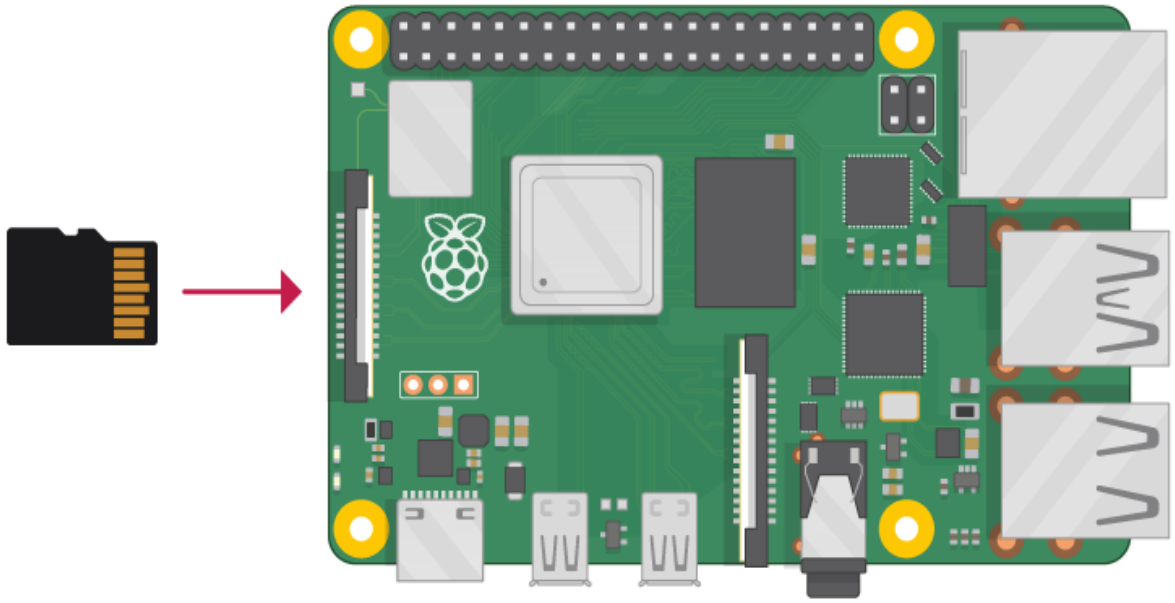
1. SD Card
2. USB Port for Keyboard & Mouse etc.
3. Power
4. Audio Jack for Sound
5. Network
6. HDMI Slots
7. Camera
8. General Purpose Pins (Electronic Components)



CONNECT YOUR RASPBERRY PI

Now get everything connected to your Raspberry Pi. It's important to do this in the right order, so that all your components are safe.

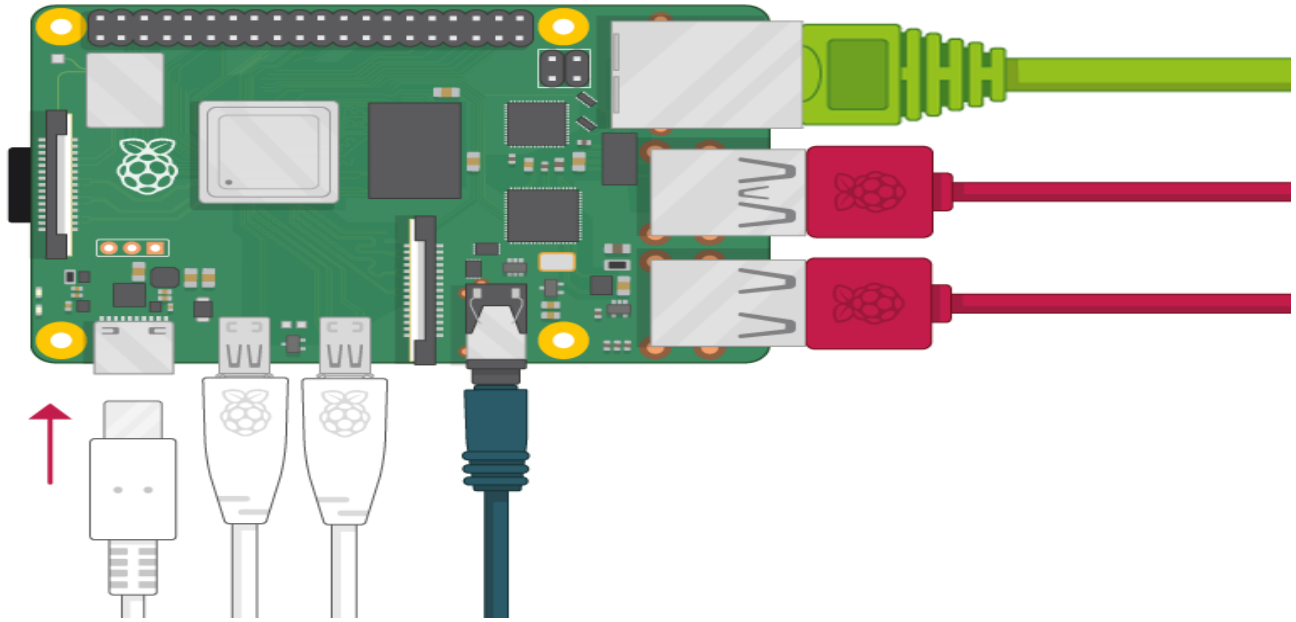
- Insert the SD card you've set up with Raspbian (via NOOBS) into the microSD card slot on the underside of your Raspberry Pi.



START UP YOUR RASPBERRY PI

Your Raspberry Pi doesn't have a power switch: as soon as you connect it to a power outlet, it will turn on.

1. Plug the USB power supply into a socket and connect it to your Raspberry Pi's power port.
2. You should see a red LED light up on the Raspberry Pi, which indicates that Raspberry Pi is connected to power. As it starts up (this is also called booting), you will see raspberries appear in the top left-hand corner of your screen.



APPLICATIONS

- It can make your Old TV into a smart TV. (You can play Videos, 3D Games, Music, Browse Internet and much more.
- Raspberry Pi can Act as Full HD 1080p Media Player.
- Its a Mini Computer which just cost Rs.2,350/-
- You can connect a Monitor, Keyboard and Mouse and use it as a normal computer.
- Its Graphics Capabilities is better than Apple Products

File Systems

TERMINAL COMMANDS

- ls
- The ls command lists the content of the current directory (or one that is specified). It can be used with the -l flag to display additional information (permissions, owner, group, size, date and timestamp of last edit) about each file and directory in a list format. The -a flag allows you to view files beginning with . (i.e. dotfiles).
- cd
- Using cd changes the current directory to the one specified. You can use relative (i.e. cd directoryA) or absolute (i.e. cd /home/pi/directoryA) paths.
- pwd
- The pwd command displays the name of the present working directory: on a Raspberry Pi, entering pwd will output something like /home/pi.
- mkdir
- You can use mkdir to create a new directory, e.g. mkdir newDir would create the directory newDir in the present working directory.
- rmdir
- To remove empty directories, use rmdir. So, for example, rmdir oldDir will remove the directory oldDir only if it is empty.
- rm
- The command rm removes the specified file (or recursively from a directory when used with -r). Be careful with this command: files deleted in this way are mostly gone for good!

- cp
- Using cp makes a copy of a file and places it at the specified location (this is similar to copying and pasting). For example, cp ~/fileA /home/otherUser/ would copy the file fileA from your home directory to that of the user otherUser (assuming you have permission to copy it there). This command can either take FILE FILE (cp fileA fileB), FILE DIR (cp fileA /directoryB/) or -r DIR DIR (which recursively copies the contents of directories) as arguments.
- mv
- The mv command moves a file and places it at the specified location (so where cp performs a 'copy-paste', mv performs a 'cut-paste'). The usage is similar to cp. So mv ~/fileA /home/otherUser/ would move the file fileA from your home directory to that of the user otherUser. This command can either take FILE FILE (mv fileA fileB), FILE DIR (mv fileA /directoryB/) or DIR DIR (mv /directoryB /directoryC) as arguments. This command is also useful as a method to rename files and directories after they've been created.
- touch
- The command touch sets the last modified time-stamp of the specified file(s) or creates it if it does not already exist.

- cat
- You can use cat to list the contents of file(s), e.g. cat thisFile will display the contents of thisFile. Can be used to list the contents of multiple files, i.e. cat *.txt will list the contents of all .txt files in the current directory.
- head
- The head command displays the beginning of a file. Can be used with -n to specify the number of lines to show (by default ten), or with -c to specify the number of bytes.
- tail
- The opposite of head, tail displays the end of a file. The starting point in the file can be specified either through -b for 512 byte blocks, -c for bytes, or -n for number of lines.
- chmod
- You would normally use chmod to change the permissions for a file. The chmod command can use symbols u (user that owns the file), g (the files group) , and o (other users) and the permissions r (read), w (write), and x (execute). Using chmod u+x *filename* will add execute permission for the owner of the file.

TERMINAL COMMANDS

- chown
- The chown command changes the user and/or group that owns a file. It normally needs to be run as root using sudo e.g. sudo chown pi:root *filename* will change the owner to pi and the group to root.
- ssh
- ssh denotes the secure shell. Connect to another computer using an encrypted network connection. For more details see SSH (secure shell)
- scp
- The scp command copies a file from one computer to another using ssh. For more details see SCP (secure copy)
- sudo
- The sudo command enables you to run a command as a superuser, or another user. Use sudo -s for a superuser shell. For more details see Root user / sudo
- dd
- The dd command copies a file converting the file as specified. It is often used to copy an entire disk to a single file or back again. So, for example, dd if=/dev/sdd of=backup.img will create a backup image from an SD card or USB disk drive at /dev/sdd. Make sure to use the correct drive when copying an image to the SD card as it can overwrite the entire disk.
- df
- Use df to display the disk space available and used on the mounted filesystems. Use df -h to see the output in a human-readable format using M for MBs rather than showing number of bytes.

- unzip
- The unzip command extracts the files from a compressed zip file.
- tar
- Use tar to store or extract files from a tape archive file. It can also reduce the space required by compressing the file similar to a zip file.
- To create a compressed file, use tar -cvzf *filename.tar.gz* *directory/* To extract the contents of a file, use tar -xvzf *filename.tar.gz*
- pipes
- A pipe allows the output from one command to be used as the input for another command. The pipe symbol is a vertical line |. For example, to only show the first ten entries of the ls command it can be piped through the head command ls | head
- tree
- Use the tree command to show a directory and all subdirectories and files indented as a tree structure.
- &
- Run a command in the background with &, freeing up the shell for future commands.

- wget
- Download a file from the web directly to the computer with wget. So wget https://www.raspberrypi.org/documentation/linux/usage/commands.md will download this file to your computer as commands.md
- curl
- Use curl to download or upload a file to/from a server. By default, it will output the file contents of the file to the screen.
- man
- Show the manual page for a file with man. To find out more, run man man to view the manual page of the man command.

Networking

- ping
- The ping utility is usually used to check if communication can be made with another host. It can be used with default settings by just specifying a hostname (e.g. ping raspberrypi.org) or an IP address (e.g. ping 8.8.8.8). It can specify the number of packets to send with the -c flag.
- nmap
- nmap is a network exploration and scanning tool. It can return port and OS information about a host or a range of hosts. Running just nmap will display the options available as well as example usage.
- hostname
- The hostname command displays the current hostname of the system. A privileged (super) user can set the hostname to a new one by supplying it as an argument (e.g. hostname new-host).
- ifconfig
- Use ifconfig to display the network configuration details for the interfaces on the current system when run without any arguments (i.e. ifconfig). By supplying the command with the name of an interface (e.g. eth0 or lo) you can then alter the configuration: check the manual page for more details.

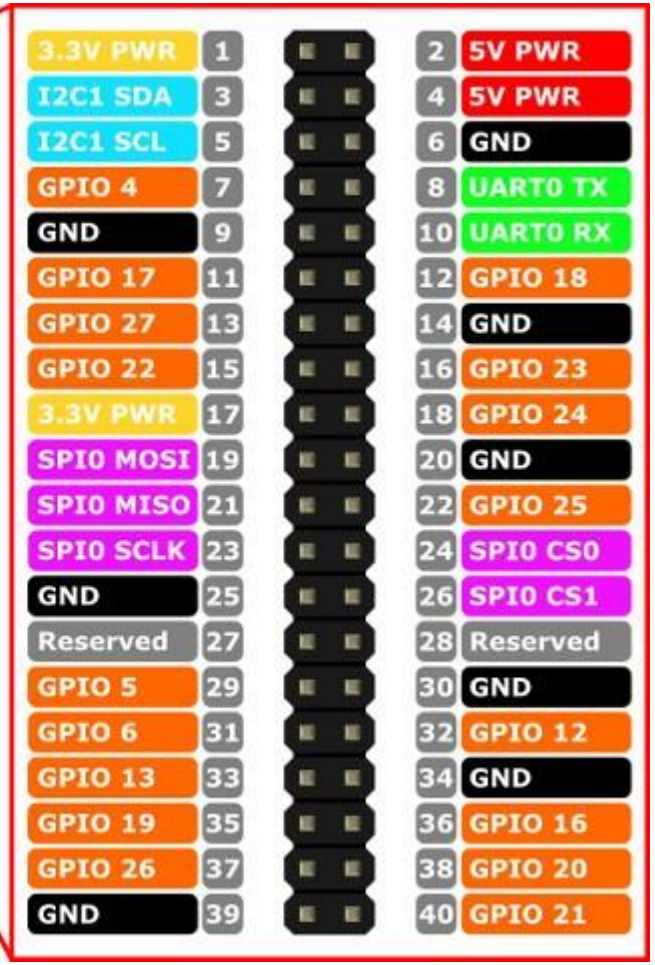
SUPER USER COMMANDS

- Sudo apt install
- Sudo apt-get install
- Sudo apt-get autoclean
- Sudo apt install tree
- Sudo tree
- Sudo apt purge tree
- Sudo apt install pip
- Sudo apt install nano
- Sudo apt install spell
- sudo apt-get install python-rpi.gpio python3-rpi.gpio
- sudo rpi-update
- Sudo passwd root
- Sudo reboot
- Sudo shutdown -h now
- https://www.linux-magazine.com/Online/Features/Convert-an-Android-Device-to-Linux

- RASPBERRY PI INTERFACING

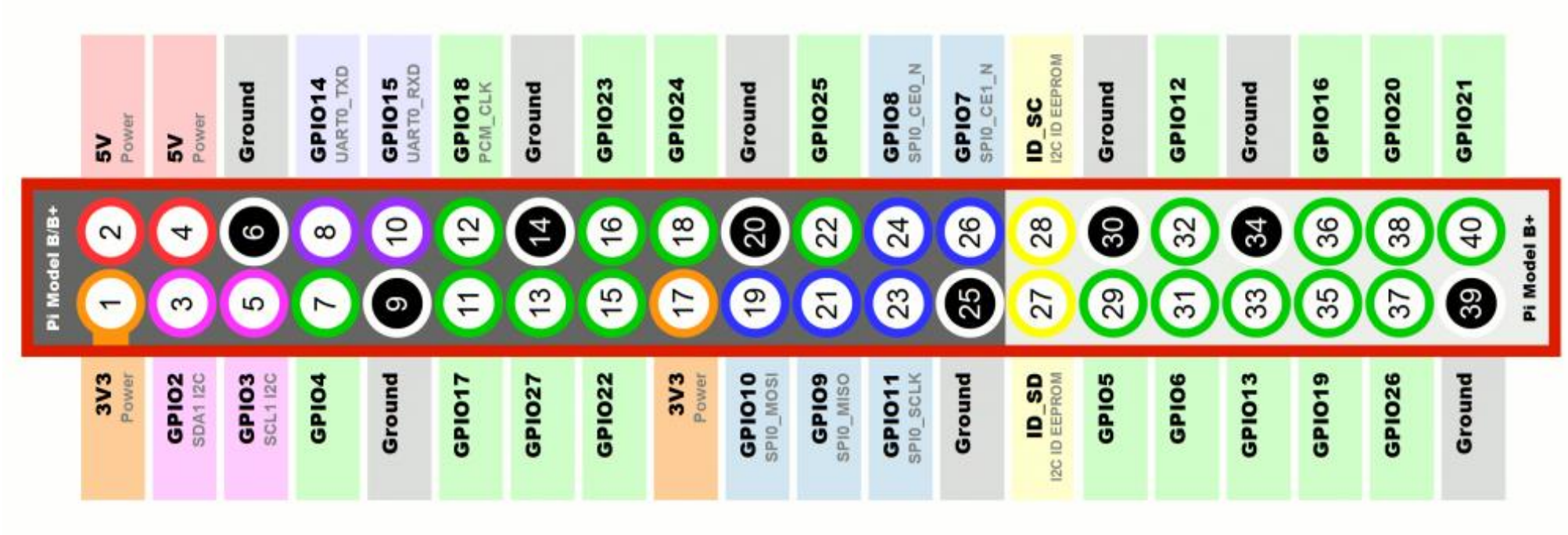
General Purpose Input/outputs, — better known as GPIO pins — using it to interface external hardware directly rather than via one of the USB ports. This is a much simpler way to The Pi is versatile and accommodating: it can be connected to anything from fundamental devices like LEDs and buzzers, to robots and smart mirrors.

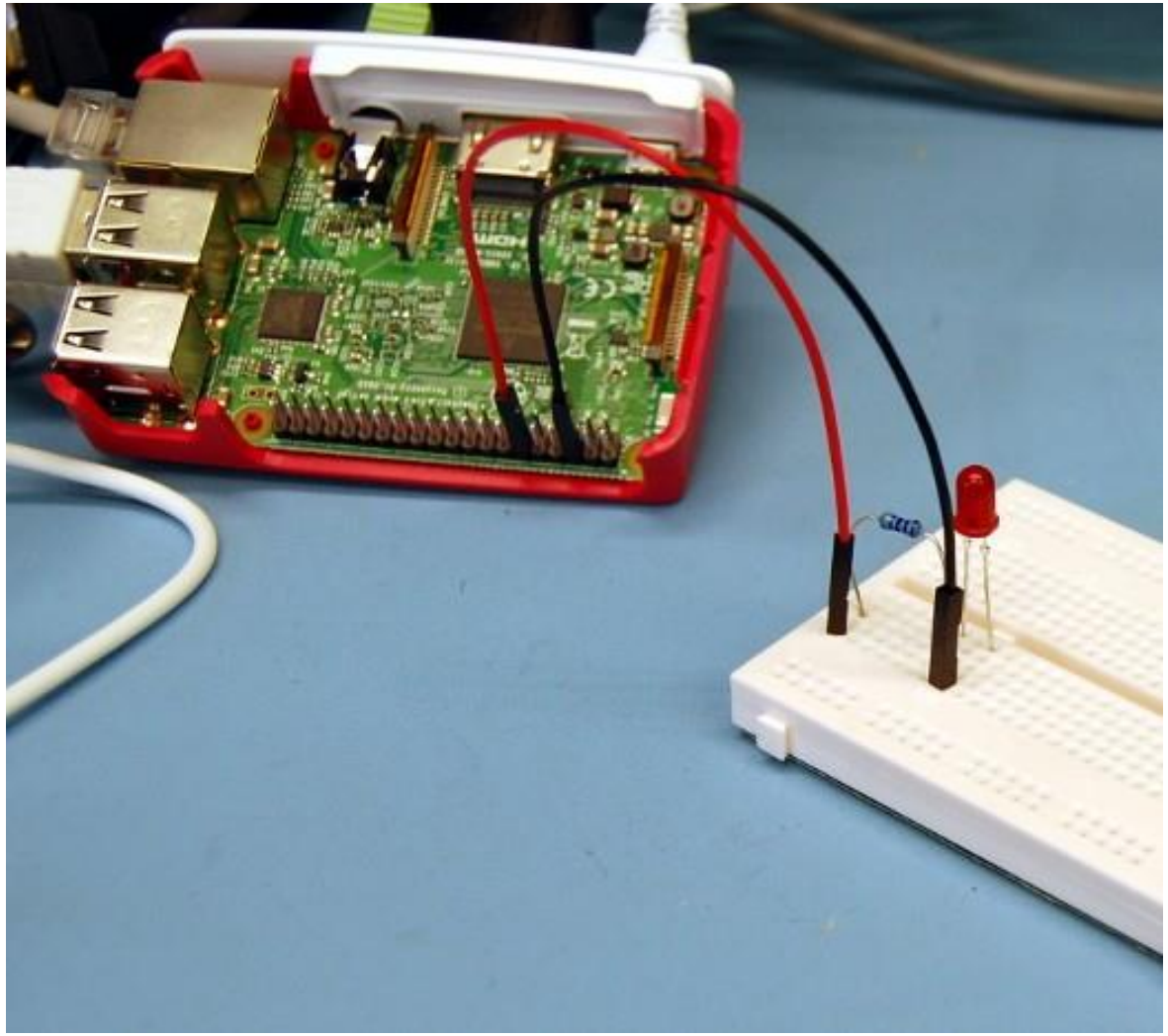
INTERFACING WITH GPIO



RASPBERRY PI GPIO PINS

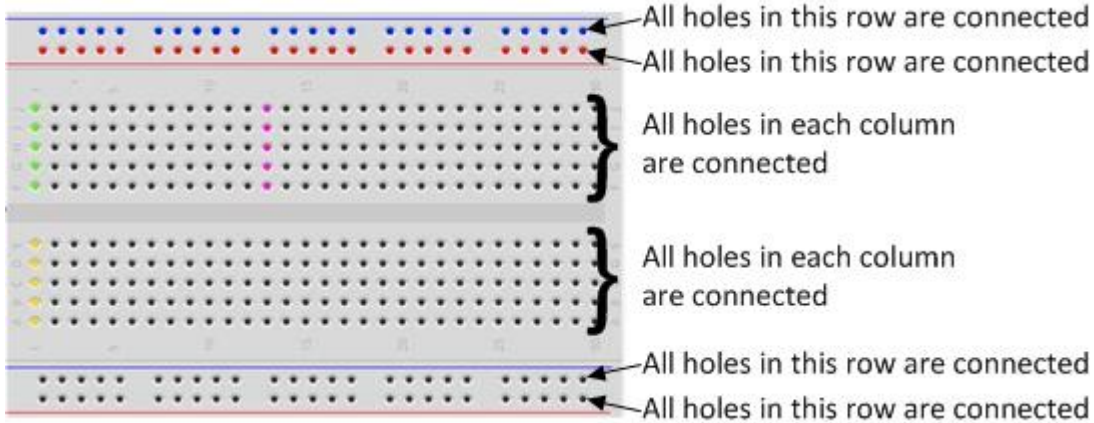
- The GPIO.BCM option means that you are referring to the pins by the "Broadcom SOC channel" number, these are the numbers after "GPIO" in the green rectangles around the outside of the diagram:





- One of the biggest selling points of the Raspberry Pi is its GPIO, or General Purpose Input/Output ports. They are the little pins sticking out of the circuit board and allow you to plug various devices into your Raspberry Pi. With a little programming, you can then control them or detect what they are doing.

BREADBOARD



- the top row of holes are all connected together – marked with red dots. And so are the second row of holes – marked with blue dots. The same goes for the two rows of holes at the bottom of the breadboard.
- In the middle, the columns of wires are connected together with a break in the middle. So, for example, all the green holes marked are connected together, but they are not connected to the yellow holes, nor the purple ones. Therefore, any wire you poke into the green holes will be connected to other wires poked into the other green holes.

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

WHY PYTHON?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

GENERAL INFORMATION ABOUT PYTHON

1. Unlike C/C++ or Java, Python statements do not end in a semicolon
2. In Python, indentation is the way you indicate the scope of a conditional, function, etc.
3. Look, no braces!
4. Python is interpretive, meaning you don't have to write programs.
5. You can just enter statements into the Python environment and they'll execute
6. For the most part, we'll be writing programs

THE PYTHON SHELL

- Because Python is interpretive, you can do simple things with the shell
- In the graphical shell on Linux, double-click on LXTerminal
- At the prompt, type `Python`
- You should have a `>>>` prompt
- Type in:

```
print("hello, world")
```

- You have written your first Python program
- Keep the shell up; we'll be using it

THE PYTHON SHELL

- This is good for simple calculations but not for real programming
- For programming, we'll use Idle
- There are two versions: Idle for Python 2.7 and Idle3 for Python 3.2
- For most of what we do, we'll have to use 2.7 because Bluetooth doesn't seem to work with 3.2
- You'll run as the "superuser" because otherwise you won't have access to the GPIO pins
- Idle will give you access to a shell but also to an IDE for writing and saving programs

1. In practice, only the simplest programs are run in the shell
2. You can create a module by going to the File->New Window menu option
3. This brings up a text editor that lets you create a Python program and run it
4. Write your first “Hello World!” program thus:
5. `print(“Hello, World!”)`

- Press F5
- It will ask you to save the file before you run it
- Save it to your home directory as HelloWorld.py
- You **must** provide the .py extension
- If you want to run it outside of the development environment simply type:

```
python HelloWorld.py
```

- Note that Linux is case sensitive

VARIABLES

- As in every language, a variable is the name of a memory location
- Python is weakly typed
- That is, you don't declare variables to be a specific type
- A variable has the type that corresponds to the value you assign to it
- Variable names begin with a letter or an underscore and can contain letters, numbers, and underscores
- Python has reserved words that you can't use as variable names

VARIABLES

- At the >>> prompt, do the following:

```
x=5

type(x)

x="this is text"

type(x)

x=5.0

type(x)
```

PRINTING

- You've already seen the print statement
- You can also print numbers with formatting
- These are identical to Java or C format specifiers

COMMENTS

- All code must contain comments that describe what it does
- In Python, lines beginning with a # sign are comment lines
- On American English keyboards, this is over the 3 key; I don't know where it is on British English keyboards

You can also have comments on the same line as a statement

```
# This entire line is a comment

x=5             # Set up loop counter
```

OPERATORS

- Arithmetic operators we will use:

  `+ - * /` addition, subtraction/negation, multiplication, division

  `%` modulus, a.k.a. remainder

  `**` exponentiation

- **precedence**: Order in which operations are computed.
  - `* / % **` have a higher precedence than `+ -`

    `1 + 3 * 4` is `13`

  - Parentheses can be used to force a certain order of evaluation.

    `(1 + 3) * 4` is `16`

When integers and reals are mixed, the result is a real number.

Example: `1 / 2.0` is `0.5`

The conversion occurs on a per-operator basis.

```
7 / 3 * 1.2 + 3 / 2
  2   * 1.2 + 3 / 2
    2.4     + 3 / 2
    2.4     +   1
          3.4
```

MATH FUNCTIONS

– Use this at the top of your program: `from math import *`

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`, `***value2***`)`	larger of two values
`min(`***value1***`, `***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

RELATIONAL OPERATORS

- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
==	equals	`1 + 1 == 2`	`True`
!=	does not equal	`3.2 != 2.5`	`True`
<	less than	`10 < 5`	`False`
>	greater than	`10 > 5`	`True`
<=	less than or equal to	`126 <= 100`	`False`
>=	greater than or equal to	`5.0 >= 5.0`	`True`

LOGICAL OPERATORS

- These operators return true or false

Operator	Example	Result
`and`	`9 != 6 and 2 < 3`	`True`
`or`	`2 == 3 or -1 < 5`	`True`
`not`	`not 7 > 0`	`False`

THE IF STATEMENT

- Syntax:

```
if <condition>:
        <statements>
```

```
x = 5
if x > 4:
        print("x is greater than 4")
print("This is not in the scope of the if")
```

THE `IF` STATEMENT

- The colon is required for the **`if`**
- Note that all statement indented one level in from the **`if`** are within it scope:

```
x = 5

if x > 4:

        print(“x is greater than 4”)

        print(“This is also in the scope of the if”)
```

THE IF/ELSE STATEMENT

```
if <condition>:
        <statements>
else:
        <statements>
```

- Note the colon following the else
- This works exactly the way you would expect

THE FOR LOOP

- This is similar to what you're used to from C or Java, but not the same
- Syntax:

```
for variableName in groupOfValues:
    <statements>
```

- variableName gives a name to each value, so you can refer to it in the statements.
- groupOfValues can be a range of integers, specified with the range function.

- Example:

```
for x in range(1, 6):
    print x, "squared is", x * x
```

RANGE

- The range function specifies a range of integers:

`range(start, stop)` - the integers between start (inclusive)

and stop (exclusive)

- It can also accept a third value specifying the change between values.

`range(start, stop, step)` - the integers between start (inclusive)

and stop (exclusive) by step

THE `WHILE` LOOP

- Executes a group of statements as long as a condition is True.
- Good for indefinite loops (repeat an unknown number of times)
- Syntax:

```
while <condition>:
    <statements>
```

- Example:

```
number = 1
while number < 200:
    print number,
    number = number * 2
```

EXERCISE

- Write a Python program to compute and display the first 16 powers of 2, starting with 1
- Do this in the Python shell

STRINGS

- String: A sequence of text characters in a program.
- Strings start and end with quotation mark " or apostrophe ' characters.
- Examples:
  "hello"
  "This is a string"
  "This, too, is a string. It can be very long!"
- A string may not span across multiple lines or contain a " character.
  "This is not
  a legal String."

  "This is not a "legal" String either."

STRINGS

- A string can represent characters by preceding them with a backslash.
  - \t tab character
  - \n new line character
  - \" quotation mark character
  - \\ backslash character

- Example: "Hello\tthere\nHow are you?"

INDEXING STRINGS

- As with other languages, you can use square brackets to index a string as if it were an array:

```
name = "Arpita Nigam"

print(name, "starts with ", name[0])
```

STRING FUNCTIONS

- `len(`***`string`***`)` - number of characters in a string
- `str.lower(`***`string`***`)` - lowercase version of a string
- `str.upper(`***`string`***`)` - uppercase version of a string
- `str.isalpha(`***`string`***`)` - True if the string has only alpha chars
- Many others: split, replace, find, format, etc.

- Note the “dot” notation: These are static methods.

BYTE ARRAYS AND STRINGS

- Strings are Unicode text and not mutable
- Byte arrays are mutable and contain raw bytes
- For example, reading Internet data from a URL gets bytes
- Convert to string:

```
cmd = response.read()

strCmd = str(cmd)
```

OTHER BUILT-IN TYPES

- tuples, lists, sets, and dictionaries
- They all allow you to group more than one item of data together under one name
- You can also search them

TUPLES

- Unchanging Sequences of Data
- Enclosed in parentheses:

```
tuple1 = (“This”, “is”, “a”, “tuple”)

print(tuple1)
```

- This prints the tuple exactly as shown

```
Print(tuple1[1])
```

- Prints “is” (without the quotes)

LISTS

- Changeable sequences of data
- Lists are created by using square brackets:

```
breakfast = [ “coffee”, “tea”, “toast”, “egg” ]
```

- You can add to a list:

```
breakfast.append(“waffles”)

breakfast.extend([“cereal”, “juice”])
```

DICTIONARIES

- Groupings of Data Indexed by Name
- Dictionaries are created using braces

```
sales = {}

sales["January"] = 10000

sales["February"] = 17000

sales["March"] = 16500
```

- The keys method of a dictionary gets you all of the keys as a list

SETS

- Sets are similar to dictionaries in Python, except that they consist of only keys with no associated values.
- Essentially, they are a collection of data with no duplicates.
- They are very useful when it comes to removing duplicate data from data collections.

WRITING PROGRAMS

- Bring up the Idle3 IDE
- The first line of your code should be this: #!/usr/bin/env python 3.1
- Write the same code you wrote for the exercise of powers of 2 using the IDE's editor
- Press the F5 key to run the program
- It will ask you to save the program. Give it a name like PowersOf2.py
- The program will run in a Python shell from the IDE
- If there are errors, the shell will tell you

WRITING FUNCTIONS

- Define a function:

```
def <function name>(<parameter list>)
```

- The function body is indented one level:

```
def computeSquare(x)
        return x * x
# Anything at this level is not part of the function
```

ERROR HANDLING

- Use try/except blocks, similar to try/catch:

```
fridge_contents = {“egg”:8, “mushroom”:20, “pepper”:3, “cheese”:2,
“tomato”:4, “milk”:13}
try:
    if fridge_contents[“orange juice”] > 3:
        print(“Sure, let’s have some juice!”)
except KeyError:
    print(“Awww, there is no orange juice.”)
```

ERROR HANDLING

- Note that you must specify the type of error
- Looking for a key in a dictionary that doesn't exist is an error
- Another useful error to know about:

```
try:
    sock = BluetoothSocket(RFCOMM)
    sock.connect((bd_addr, port))
except BluetoothError as bt
    Print("Cannot connect to host: " + str(bt))
```

USING THE GPIO PINS

- The Raspberry Pi has a 40-pin header, many of which are general-purpose I/O pins
- Include the library:

```
import RPi.GPIO as GPIO
```

- Set up to use the pins:

```
GPIO.setmode(GPIO.BOARD)
```


58

PROGRAMMING EXERCISE

- Write a Python program that blinks an LED at a rate of 1 second on, one second off
- To do this, you'll need to use the idle3 environment running as the superuser:

```
sudo idle3
```

PYTHON FILE I/O

- You can read and write text files in Python much as you can in other languages, and with a similar syntax.
- To open a file for reading:

```
try:
    configFile = open(configName, "r")
except IOError as err:
    print("could not open file: " + str(err))
```

PYTHON FILE I/O

- To read from a file:

```
while 1:
  line = configFile.readline()
  if len(line) == 0:
      break
```

- You can also read all lines from a file into a set, then iterate over the set:

```
lines = file.readlines()
for line in lines:
    print(line)
file.close()
```

PYTHON FILE I/O

- Writing to a text file

```
file=open('test.txt',"w")
file.write("This is how you create a new text file")
file.close()
```

USING THE GPIO PINS

- This is from my home-control code:

```
LAMP = 22

MOTION = 23

GPIO.setup(LAMP, GPIO.OUT)   # For turning on the lamp

GPIO.setup(MOTION, GPIO.IN)  # For reading the motion sensor
```

- Like the Arduino, we must set up the pins for input or output

USING THE GPIO PINS

- Reading from a GPIO pin:

```
# If we detect motion, print that.

if GPIO.input(MOTION):

    print( "Motion detected")
```

USING THE GPIO PINS

- Output to GPIO:

```
if cmd=='LAMPON':
    cmdlist["LAMPSTATUS"] = True;
    GPIO.output(LAMP, True)     # turn on the light
```

GPIO IN SCRATCH

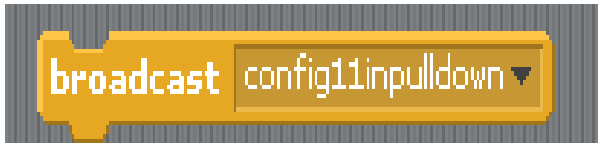
- For instance, to configure GPIO pin 4 as an output and turn it on, you create the two following broadcasts:

- As always, you can assemble this text with normal join, pick, or list-handling blocks. For example, if foo = 17, then

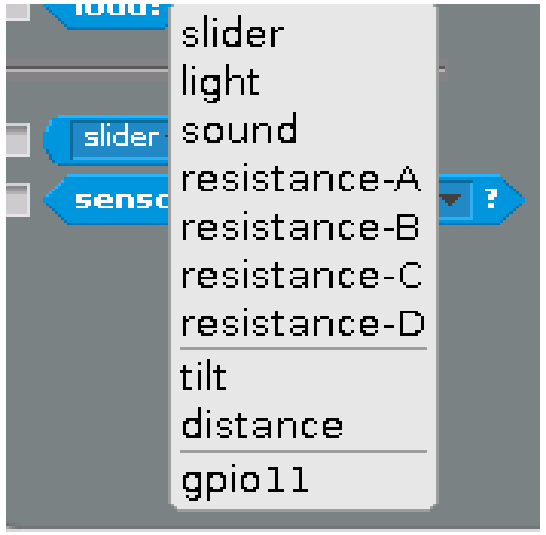
- would broadcast gpio17on and thus set the GPIO pin number 17 (under the BCM numbering - not the physical or wiringPi numbers!) to on.

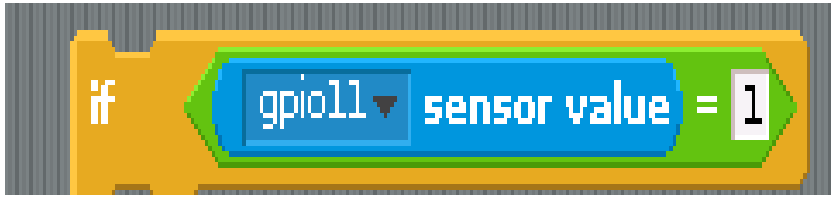
GPIO IN SCRATCH

•We can set the pull-up resistor to float with 'inpullnone' or 'inputpullnone'

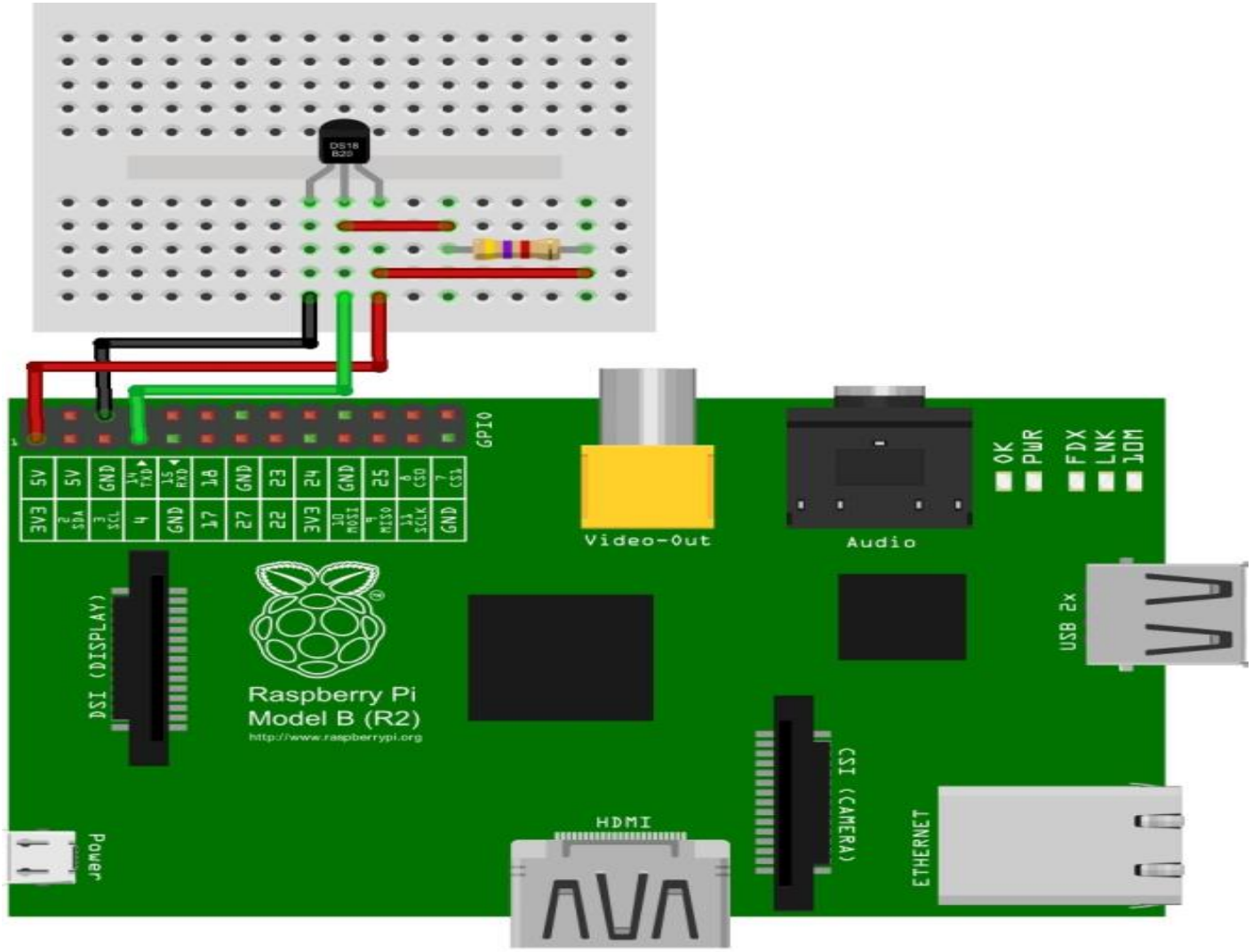


•Pins set to be inputs are connected to the Scratch sensor variable system, and so they appear in the list of possible values in the sensor blocks:





INTERFACING TEMPERATURE SENSOR WITH RASPBERRY PI



PYTHON CODE FOR SENSOR INTERFACING

- import os # import os module
- import glob # import glob module
- import time # import time module
- os.system('modprobe w1-gpio')
- os.system('modprobe w1-therm')
- base_dir = '/sys/bus/w1/devices/'
- device_folder = glob.glob(base_dir + '28*')[0]
- device_file = device_folder + '/w1_slave'
- def read_temp_raw():
- f = open(device_file, 'r')
- lines = f.readlines() # read the device details
- f.close()

PYTHON CODE FOR SENSOR INTERFACING

```
return lines
def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':                # ignore first line
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')                    # find temperature in the details
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0    # convert to Celsius
        temp_f = temp_c * 9.0 / 5.0 + 32.0      # convert to Fahrenheit
        return temp_c, temp_f
while True:
    print(read_temp())                          # Print temperature
    time.sleep(1)
```

DCS

THANK YOU

JUST BE
ADAPTIVE
AND
FOCUSED