## What you've learned

**Variable** A variable stores a value. For example in our program we created three variables, `x`, `y` and `z`, which stored the values `10`, `50` and `12` respectively. You can change the values `10`, `50` and `12` to any number that you want. Variables can be reused. When we used `setPos()` on line 6 we were reusing the variables `x`, `y` and `z`.

**Function** A function is a reusable piece of code that performs a specific purpose. For example we used the pre-written function `setPos()` on line 6 to change the position of the player in the game world.

**Function arguments** Some functions need to be given data in order to work, this data is called an argument. For example on line 6 we gave the argument `x`, `y` and `z` to the `setPos()` in order to tell it where to teleport the player to.

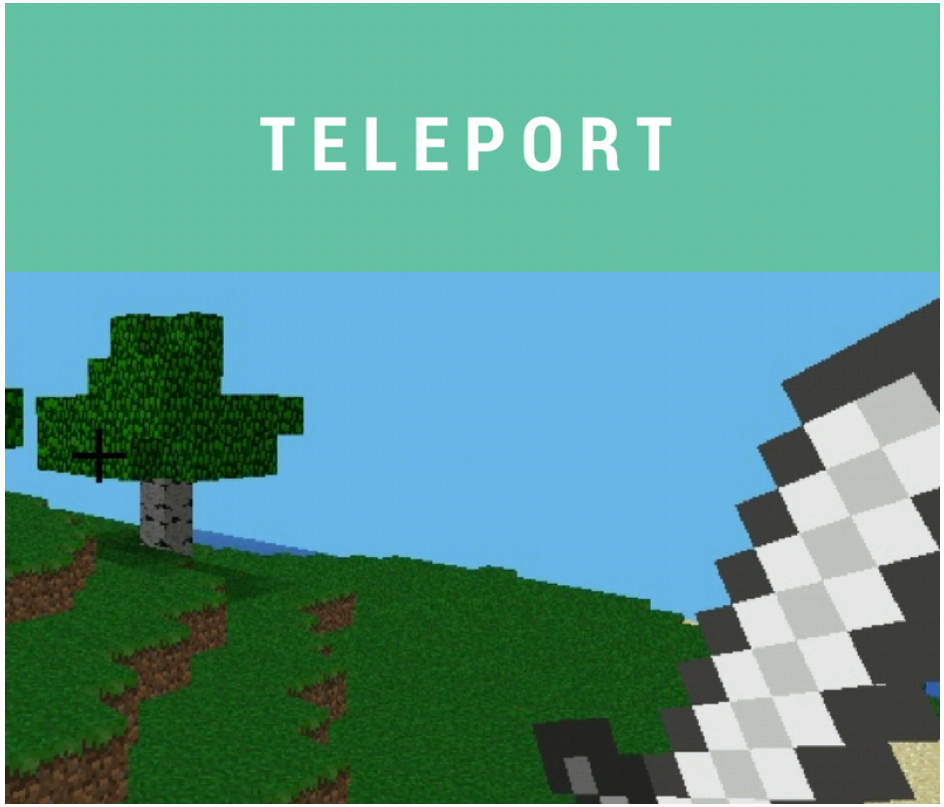
**Minecraft Pi API** An **API** is a collection of pre-written functions that allow you to connect your Python code to another program. In this guide you have learned how to connect your Python programs to the Minecraft Pi API, which contains functions to control Minecraft Pi games.

## Your Turn!

Here are some ideas to change the code and extend your teleport code.

- What happens when you change the values of the `x`, `y` and `z` variables?
- What happens when you one of the values of `x`, `y` and `z` variables to a negative number?






# Teleport

Programming with Python on the Raspberry Pi is a powerful and fun way to modify Minecraft games. With a few lines of code you can take control of the player and the every block in the world.

In this first worksheet you will use some basics of Python programming with Minecraft Pi. You'll learn how to teleport the player to a new position on the map using variables in Python.

Before you begin you'll need to create and save a Python program file. You'll also need to copy the set of Minecraft instructions that allow Minecraft to connect to Python. The next page includes instructions that show you how to do this.

# Setting Up

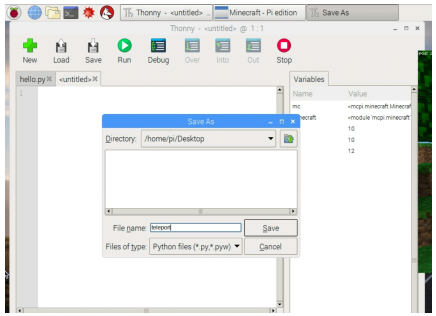
1. **Turn on the Raspberry Pi**

2. **Open Minecraft:**
   To open Minecraft Pi click on the Raspberry Pi start menu on the upper left hand corner. Go to **Games** -> **Minecraft Pi**.

   Start a game and create a new world.

   **Alert:** Press tab in Minecraft to release the mouse

3. **Open Thonny:**
   Thonny is used to run Python programs. Go to your start menu again -> **Programming** -> **Thonny (Simple Mode)**.




4. **Save and Run:**
   In Thonny, click the "New" icon to start a blank file, then "Save" icon.

   Save your file as "teleport.py" on your Desktop.

   Type the Python code below into Thonny. Click the "Run" icon to run your code.

# Code

## Import the API

Every Minecraft Pi program that you write in Python requires these two lines of code. The first line imports the commands that allow you to interact with a Minecraft game using Python. The second line creates a connection to the game.

```
1 from mcpi.minecraft import Minecraft
2 mc = Minecraft.create()
```

## Set the Variables

The player's position in Minecraft is represented using coordinates. Here we have created three variables to represent the player's position. A variable stores a value, in this case the variables `x`, `y` and `z` store the values `10`, `50` and `12` respectively.

```
3 x = 10
4 y = 50
5 z = 12
```

## Teleport the player

The last line teleports the player to a new position in the game. The `setPos()` function uses three number values, known as

arguments, to change the player's position. In this case we're using the values of the `x`, `y` and `z` variables that we set earlier.

```
6 mc.player.setPos(x, y, z)
```

## What you've learned

**`while` loop** A `while` loop repeats a section of Python code. In this example the while loop repeats lines 5–11. The `True` part of the `while` loop means that it will repeat forever or until the user terminates the program.

**`time`** Using time in Python allows us to use functions that control `time`. For example the program that you've created uses the `time.sleep()` function to pause the Python program for a short type before continuing.

**`getPos()`** The `getPos()` function is part of the Minecraft Pi API and allows us to find the coordinates of the player in the game world.

**`setBlock()`** The `setBlock()` function in the Minecraft Pi API allows us to create blocks in the Minecraft Pi world. It takes four arguments, the first three of which are coordinates and the fourth argument is the type of block that we want to create.

## Your Turn!

Here are some suggestions to extend your code and make it do different things. Try anything you think of too!

- Change the block type that is placed. To do this change the value of the `block` variable on line 9. Some examples include melons (value 103), gold (value 41) and water (value 8).




# Flower Path

With Python you can achieve things that aren't normally possible in a Minecraft Pi game. In this exercise you'll create a trail of flowers that follows the player wherever they go.

You'll learn a number of new things in Python and the Minecraft Pi API. This code can be easily modified to change the flowers to any other block type including gold and TNT.

# Code

## Import the API

This is the code that you always use to connect your code to a Minecraft Pi game.

```
1 from mcpi.minecraft import Minecraft
2 mc = Minecraft.create()
```

## Import time

This statement allows us to use time commands in our programs. For example we will use `time.sleep()` in our program which makes our program wait a number of seconds before continuing to the next line.

```
3 import time
```

## While loop and get player position

A `while` loop repeats a block of indented code. In this case it will repeat lines 5-11. The `True` part of the loop means that this loop will run forever until the user terminates the program with `ctrl+c`. Lines 5-8 find the player's position and then stores it in the `x`, `y` and `z` variables.

```
4 while True:
5     pos = mc.player.getPos()
6     x = pos.x
7     y = pos.y
8     z = pos.z
```

## Set the block type

Every block type in the Minecraft world has an associated number. For example air is 0, lava is 12 and melon is 103. In order to place flowers we store the flower block's number, 38 in the `block` variable.

```
9     block = 38
```

## Set the block

The `setBlock()` function creates a block in the Minecraft world. It requires coordinates and a block type. Here the program is using the variables created on lines 6-9.

```
10    mc.setBlock(x, y, z, block)
```

## Slow the loop

The last line of the loop contains `sleep()`, which makes the program wait 0.2 seconds before looping again. We do this so that the loop doesn't work too quickly and use up too much processing power.

```
11    time.sleep(0.2)
```

## What you've learned

**Maths operators** Addition, subtraction, multiplication and division are all possible in Python. Using them in Python allows you to change the values of variables or arguments. For example we used addition on lines 9–11 to add to the values of `x`, `y` and `z` and store the result in new variables. We also used addition and subtraction to modify the values of arguments on lines 16-20.

**`setBlocks()`** The `setBlocks()` function of the Minecraft Pi API creates a cuboid of blocks between two sets of co-ordinates. These co-ordinates are provided as arguments. In the example we used six variables for these arguments. The seventh argument is the type of block that will be created.

## Your Turn!

Here are some suggestions to extend your code and make it do different things. Try anything you think of too!

- Change the block types of the blocks being placed. A popular block type for the inside is lava (value 11) with an surrounding of glass (value 20).
- Make a swimming pool. Change the values on the maths operators used in the `setBlocks()` function on lines 16-20 so that there is no top roof created.




# Warehouse

When you're building in Minecraft you can spend ages on a relatively simple structure, such as the walls of your house. Wouldn't it be nice if you could save time? Well, you can do that with only a few lines of Python.

In this guide you'll learn how to use maths operators in Python to create the basic structure for a house. You'll create a hollow box of block, which you can build upon to create your own ideas with more speed.

You can have loads of fun adapting this code. Come up with your own ideas and try changing the block types or combining it with other bits of code from other exercises.

# Code

## Import the API and set variables

As usual we import the API and connect to the game. The `block` and `air` variables store block types, which we will use later. Block type 4 is cobblestone.

```
1  from mcpi.minecraft import Minecraft
2  mc = Minecraft.create()
3  block = 4
4  air = 0
```

## Set variables

The `x`, `y` and `z` variables represent the bottom left hand corner of a 3D rectangle, known as a cuboid. The variables `x2`, `y2` and `z2` add to the `x`, `y` and `z` variables and store the new values. The `+` operator is used for addition. Likewise `–` can be used for subtraction.

```
5  x = 10
6  y = 11
7  z = 12
8  x2 = x + 10
9  y2 = y + 5
10 z2 = z + 8
```

## Create blocks

Similar to `setBlock()`, the `setBlocks()` function creates blocks in a Minecraft game. It creates a cuboid (3D rectangle) between two points. These points have been set using the variables that we created on lines 5–10. We also provide the block variable to say what type of blocks should be created in the cuboid.

```
11 mc.setBlocks(
12   x, y, z,
13   x2, y2, z2,
14   block
15 )
```

## Create air blocks

We once again call the `setBlocks` function. This time we use addition and subtraction to create a cuboid made of air within the cuboid we created on lines 11-15. The `air` variable sets the cuboid to air, which effectively hollows the original cuboid, leaving a ceiling, floor and walls that are one block thick.

```
16 mc.setBlocks(
17   x + 1, y + 1, z + 1,
18   x2 - 1, y2 - 1, z2 - 1,
19   air
20 )
```

## What you've learned

**`input()`** The `input()` function in Python allows the user to input strings (text). The string is then stored in a variable and can be reused in the rest of the program. For example in our program we created a variable called `chatMsg`.

**Strings** Strings are a data type in Python. To you they're the same thing as text. You can identify a string by the speech marks that surround it. For example we have used a string, `"Enter a message: "` on lines 3 and 6 and `"/exit"` on line 4. The user input of `chatMsg` is also stored as a string.

**Not equal to (`!=`)** The not equal to comparator checks that two values are not the same. As long as the `chatMsg` variable is not equal to `"/exit"`, the `while` loop on line 4 will repeat. If the `chatMsg` variable is equal to `"/exit"`, the loop will stop.

**`postToChat()`** The `postToChat()` function takes a string as an argument and displays it on Minecraft's chat.

## Your Turn!

Here are some suggestions to extend your code and make it do different things. Try anything you think of too!

- Add a user name to the chat. Add an extra line before line 3 that uses `input()` to ask the user's name and store it in a `userName` variable. Then change `postToChat()` to `mc.postToChat(userName + ": " + chatMsg)`




# Chat

Minecraft Pi Edition has many hidden features that you can only access using Python code. One of those hidden features is the chat feature.

In this guide you'll learn how to use strings in Python to create a chat program. If you're playing with other players on multi-player, this program will be very useful for communicating.

This program works best if you're playing over a network with other people. Can you work out how to add a user-name to the chat?

# Code

## Import the API

As usual we import the API and create a connection to the game.

```
1 from mcpi.minecraft import Minecraft
2 mc = Minecraft.create()
```

## Get the user input

The `input()` method takes user input from the command line. The string (text) inside the brackets is printed to the command line and whatever the user inputs is returned and stored in the `chatMsg` variable. When programming, a “string" is the correct term for text.

```
3 chatMsg = input("Enter a message: ")
```

## While loop with a condition

Once again we’re using a while loop to repeat some code. The difference here is that the loop will only repeat if the value of the `chatMsg` variable is not equal to `"/exit"`. If the value of `chatMsg` is equal to `"/exit"`, the loop will no longer repeat and the program will finish.

```
4 while chatMsg != "/exit":
```

## Post the message

The `postToChat()` function displays a string (text) on the Minecraft Pi in-game chat. In this case we’re displaying the value of `chatMsg`.

```
5     mc.postToChat(chatMsg)
```

## Get another user message

At the end of the loop we get the next chat message from the user and store it in the `chatMsg` variable. The code on line 3 is outside the loop therefore we need to write it again within the loop in order for our program to work properly.

```
6     chatMsg = input("Enter a message: ")
```

## What you've learned

**`getBlock()`** The `getBlock()` function finds the type of a block at certain co-ordinates. The co-ordinates are given as arguments to the function, for example in this program we use the `x`, `y` and `z` variables as these arguments. The function returns the block type at those co-ordinates.

**`if` statement** An `if` statement will only run a section of code when a condition is `True`. When the condition is `False`, the section of code will not run. For example in our program the block below the player will only turn to ice if it is water.

**Equal to (==)** The equal to operator checks whether one value is the same as the other. If they are the same it will evaluate to `True`, if they are not it will evaluate to `False`. In our program we use an equal to operator with an `if` statement to check whether the block below the player is water.

## Your Turn!

Here are some suggestions to extend your code and make it do different things. Try anything you think of too!

- Change the block type that is changed. Try changing it to lava (block value `10`).
- Check that the block below the player is not air (block value `0`). Use the not equal to operator (`!=`). This will mean that every block the player passes over, except air will be changed.




# Freeze

One of the benefits of programming is that it can automate decision making. This saves us time and allows us to create rules that our programs must follow.

In this guide we'll use Python code to freeze any blocks of water below the player. By using decision making in Python, we can avoid freezing any other blocks than water.

By making small changes to the code you can change the effect of the program quite drastically. For example you could make the program change grass to gold. Try coming up with your own ideas!

# Code

## Import the API

As usual we import the API and make a connection to the game. We also import `time`.

```
1  from mcpi.minecraft import Minecraft
2  mc = Minecraft.create()
3  import time
```

## Start the loop and get the player's position

As we need to repeat the code constantly we use an infinite `while` loop. We slow down the speed that the loop repeats with `time.sleep()`. On lines 6–9 we find out the player's position and store it in the `x`, `y` and `z` variables.

```
4  while True:
5      time.sleep(0.2)
6      pos = mc.player.getPos()
7      x = pos.x
8      y = pos.y
9      z = pos.z
```

## Get the block below the player

To find the type of a block in the Minecraft game we use the `getBlock()` function. It uses co-ordinates and returns the block's type at those co-ordinates. To find the block below the player we subtract `1` from the `y` variable.

```
10     blockBelow = mc.getBlock(x, y - 1, z)
```

## Create block variables

On lines 11–12 we create variables to store the block types of water and ice, which we will use later.

```
11     water = 9
12     ice = 79
```

## Check the block below the player

To check the block below the player we use an `if` statement. An `if` statement only executes some code when something is true. In this case it checks whether the block below the player (stored in the blockBelow variable) is water. If the block below the player is water it will set the block below the player to ice.

```
13     if blockBelow == water:
14         mc.setBlock(x, y - 1, z, ice)
```

## What you've learned

**lists** A `list` is a datatype that can store several values. Think of it like a shopping list, you have a number of items in an order. The `hits` variable is an example of a list in our program as it stores all of the block hits that the player made in 60 seconds.

**`pollBlockHits()`** The `pollBlockHits()` function returns a list of all of the block hits that the player has made with a sword since the start of the program. The list contains co-ordinates of the blocks that the player has hit. Only right-clicks with a sword will be returned. The co-ordinates are accessed on lines 8-10.

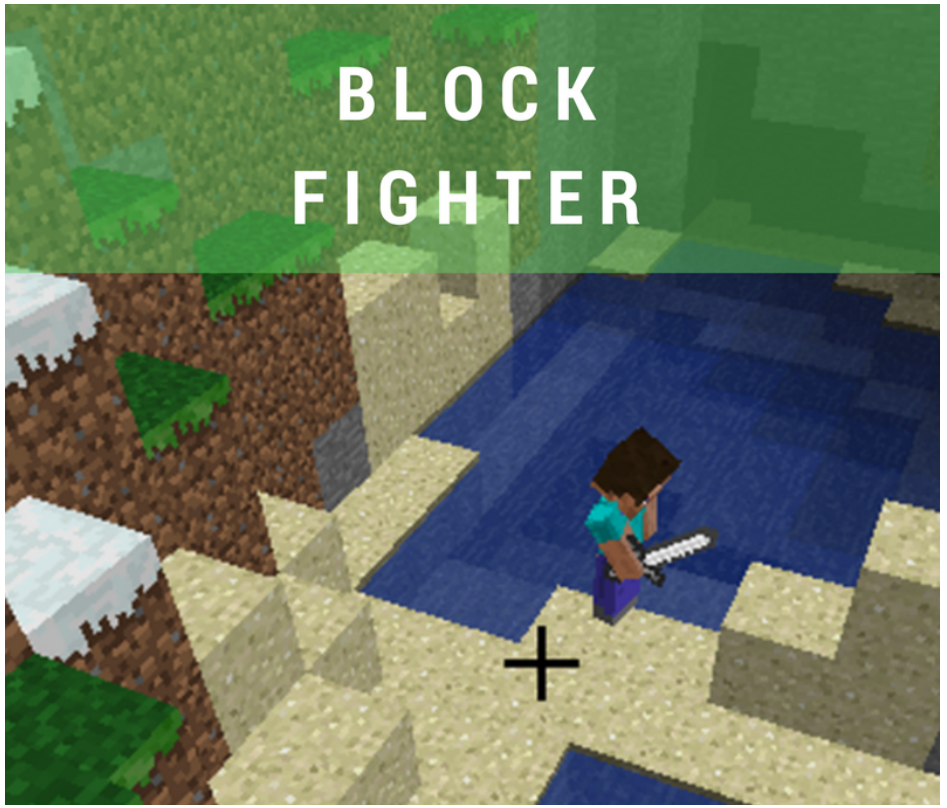
**`for` loops** A `for` loop will repeat a section of code a number of times. In the example the `for` loop will repeat for every item in the `hits` list. A `for` loop stores the value of each item in a list in a variable that changes each time it runs. For example in our code the `hit` will hold the value of the current item in the `hits` list each time its run. This variable can be used in the body of the loop.

## Your Turn!

Here are some suggestions to extend your code and make it do different things. Try anything you think of too!

- Make a magic wand. Rearrange the code so that every time the player hits a block with their sword, it is magically transformed into a rare block type.






# Block Fighter

Using the Minecraft Pi API you can create your own mini-games. This means you can build upon the existing game to create your own ideas quickly without having to start from scratch.

In this guide we'll learn how to use the for loop to create a mini-game that gives the player points for the different blocks they hit/right-click with a sword. By using time in the Python program the player has a time limit in which they must hit the blocks.

Try adapting the code when you're done. For example you could make your sword place rare blocks.

# Code

## Import the API

As usual we import the API and connect to the game.

```
1  from mcpi.minecraft import Minecraft
2  mc = Minecraft.create()
```

## Wait a bit

We wait for 60 seconds in order to give the player time to hit some blocks with a sword. The `points` variable will store the player's score.

```
3  import time
4  time.sleep(60)
5  points = 0
```

## Record the block hits

The `pollBlockHits()` function returns a list of block hits. A `list` contains several a values that can be stored using a single variable. This means you can store several values without needing a separate variable for each.

```
6  hits = mc.events.pollBlockHits()
```

## Loop through every block hit

A `for` loop is a type of loop that repeats for each item in a `list`. In this case it loops through each item in the `hits` list and stores each value in the `hit` variable, one at a time. The code inside the loop gets the co-ordinates that the player hit and finds the block type at those co-ordinates. The value of the block type is then added to the `points` variable.

```
7  for hit in hits:
8      x = hit.pos.x
9      y = hit.pos.y
10     z = hit.pos.z
11     points = points + mc.getBlock(x, y, z)
```

## Post the points to the chat

Once the for loop has ended we display the number of points that the player scored to chat in the Minecraft game.

```
12 mc.postToChat(str(points) + " points.")
```