# Girls' Programming Network

*Bop It with micro:bits!*

# This project was created by GPN Australia for GPN sites all around Australia!

## This workbook and related materials were created by tutors at:

Sydney, Canberra



Girls' Programming Network

*If you see any of the following tutors don't forget to thank them!!*

# Part 0: Setting up

## Task 0.1: micro:bits and pieces

**Let's set up the micro:bit for programming today! You should have:**

- **1 micro:bit chip**
- **1 USB cable**

1. Connect the small end of the USB cord to the middle port of the micro:bit
2. Connect the big end of the cord to your computer
3. Go to **python.microbit.org**

## Task 0.2: Micro playground

First we're going to play around with the displays on **microbit.org** and test them on our micro:bits.

1. Make sure `from microbit import *` is at the top of your code.
2. Change the code under the `while True:` loop to display a duck and scroll your name instead
3. Click the **'Send to micro:bit'** button to try this out. Then follow the steps on the screen.
4. Try this out with other words and pictures.

### Hint

Don't forget you have cheat sheets to help you code!
Remember to indent the code below the while loop!

## ☑ CHECKPOINT ☑

**If you can tick all of these off you can go to Part 1:**

☐ You are have connected your micro:bit to the computer

☐ You can display different pictures and words

# Today's Project Plan - Bop It

**We're going to make a Bop It game!**
**It will prompt the user to press A or B to get points! Get as many points as you can in the time limit.**

**1**   Start off the game by showing a starting image!

**2**   List your actions, and choose a random action to be the first move!

**3**   Display different images on the screen depending on what action you chose!

**4**   Add a loop to make it choose and display actions over and over again!

**5**   Make the game wait for you to complete the action. Get a smiley and new move when you're correct!

**6**   Add scores to the game and show the final score at the end of the game!
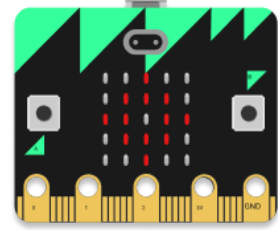
# + more

**Once your base game works add cool extensions!**
There is extension sounds, making your own buttons out of foil, using radio communication to make multiplayer games and many more!!

# Part 1: Ready! Set! Go!

> ## We need to know the game is starting!
>
> Display an image that tells the player the game is starting!

## Task 1.1: Name your file!

Now you're used to working with your micro:bit, let's start working on the project!

1. On your Python Editor on **microbit.org**. At the top of the page, edit your project name to be 'bop_it'.

2. Delete all the code except for `from microbit import *`

3. At the top of the file, add a new line (above the import line) and use a comment to write your name.

### Hint

Remember comments start with a `#`
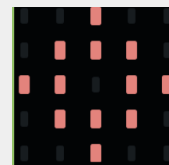`# Comments don't actually do anything - they are just notes!`

## Task 1.2: Starting your game

To show that the game is starting, let's show a target image for 1 second!

1. In a new line below the import statement, use `display.show()` to show a target. (called `Image.TARGET`)

2. Make the program `sleep` for `1000` milliseconds.

3. Then `clear` the display with `display.clear()`

## ☑ CHECKPOINT ☑

**If you can tick all of these off you can go to Part 2:**

☐ Your program shows a target at the start of the game for 1 second and then clears the display.

☐ You tried it on your real life micro:bit

# Part 2: Choosing a move

> ## Our game is about doing random actions!
>
> Let's start by chosing the first move!
> What will be Button A or B?

## Task 2.1:  Making a list of actions

**We need to make a list of actions to refer to later.**

1. At the end of your code, create a **list** called **actions**.

2. Inside the **list** store the two actions **"press a"** and **"press b"**.

### Hint

Remember a **list**  looks like this:
```
fave_foods = ["pizza", "curry", "nutella", "omelette"]
```

## Task 2.2: Get random

**To randomly select actions in our game we'll need to import a special library.**

Underneath **from microbit import \*,** add a new line of code that says **import random**

## Task 2.3: Selecting the next action

**Now we'll use the library to choose a move from our list of actions.**

1. On a new line after our list of **actions**, create a variable called **action**.

2. Choose a random action from the list of **actions** and assign it to **action**.

3. Then **print** the **action** so we can see what it is.

   *It will print to the serial under the simulator micro:bit (make sure you click "show serial to see it!)*

## Task 2.4: Check that it works!

**Now you need to run your program a few times to check that it is working!**

1. Run your code multiple times. See what action it prints out.

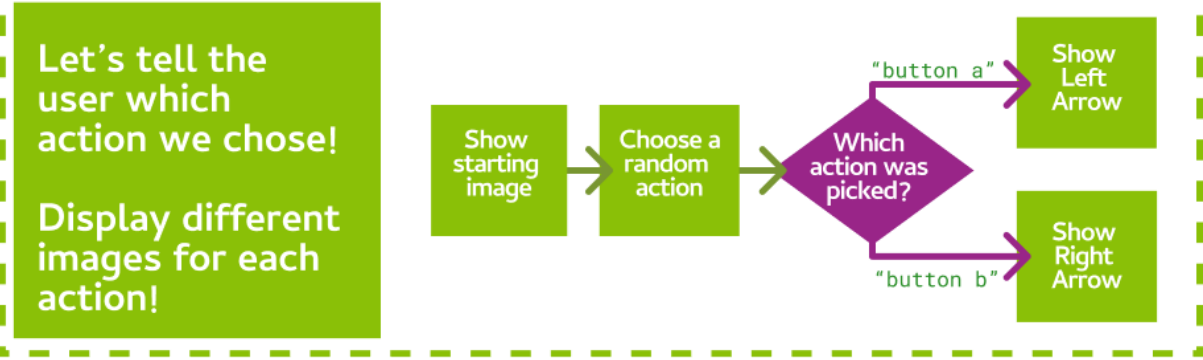**Do you get different actions?** You might get the same one a few times in a row.

## ☑ CHECKPOINT ☑

**If you can tick all of these off you can go to Part 3:**

☐ You have a list of `actions`

☐ You choose an action using `random`

☐ You have the next `action` stored in a variable

☐ You have a print statement that `prints` out the `action`

# Part 3: Light it up!

Let's tell the user which action we chose!

Display different images for each action!

Show starting image → Choose a random action → Which action was picked?

"button a" → Show Left Arrow

"button b" → Show Right Arrow

## Task 3.1: What's your action?

In part 2, you made two **actions** your game can choose from.

Do you remember what they were called?

**Write their names down below:**

1) ...................................................

2) ...................................................

## Task 3.2:

We want to point to the button the player should press.

**Which action will we show for each of these images?**

**Image.ARROW_W**

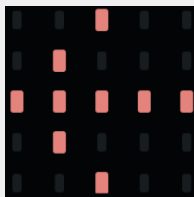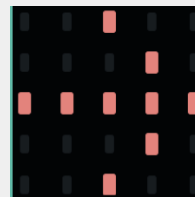**Action:** ............................

**Image.ARROW_E**

**Action:** ............................

## Task 3.3: Giving the first action a picture

**Let's check what action was selected and display a picture!**

1. At the bottom of your code, create an `if` statement that checks whether the `action` the computer chose is `"press a"`.

2. Inside the `if` statement, use `display.show()` to show the arrow that points to button **A.** Make sure it's indented.

## Task 3.4: Giving the second action a picture

**Now we'll do the same for the other action**

1. Create another `if` statement underneath the previous one that checks whether `"press b"` is the action.

2. Inside this `if` statement, display an arrow that points to button **B**.

## Task 3.5: Testing time!

**Run your code!**

1. Check the terminal to see which `action` was selected.

2. Does it `display` the correct picture for the randomly chosen action?

3. Run your program multiple times to check both actions!

## ☑ CHECKPOINT ☑

**If you can tick all of these off you can go to Part 4.**

☐ When you run the program, it shows **one** of the pictures below

Picture 1: 
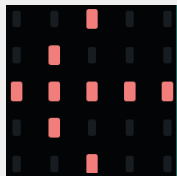
Picture 2: 
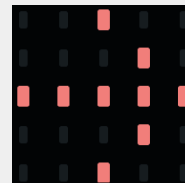
☐ If you run the program multiple times, it shows the **other** picture sometimes. (This might take a few goes)

## ★ BONUS 3.6: Choose your own pictures! ★

**Waiting for the next lecture? Try adding this bonus feature!!**

Instead of showing left and right arrows, let's choose our own pictures!
- Replace the code that shows the **left arrow** with `Image.SQUARE`
- Replace the code that shows the **right arrow** with `Image.HEART`

What do you see when you run the program? What if you rerun the program a few times?

**Find more images on the *micro:bit Image Cheat Sheet*:** http://bit.ly/images-microbit

# Part 4: The more actions the merrier!

## One action is not enough for a game!
### Let's make it play on a loop and show new actions for 10 seconds!
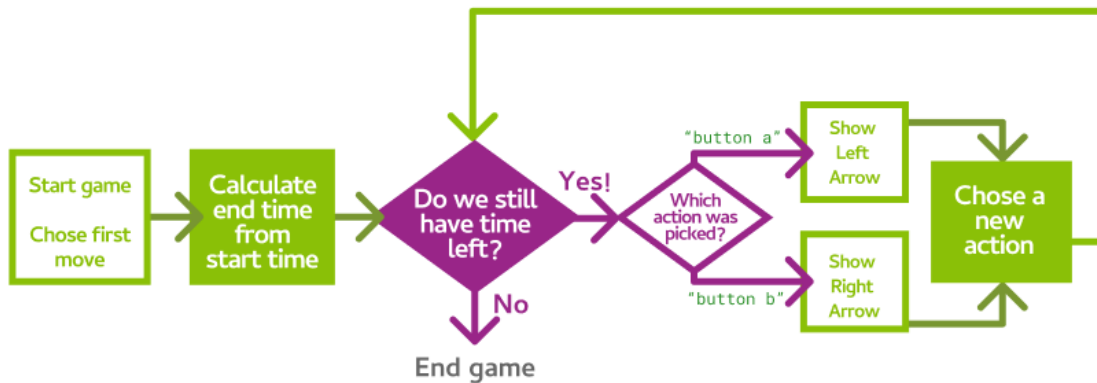


## Task 4.1:  Looping for 10 seconds

**To know when to stop the game, we need to know when it started!**

1. In a new line after you randomly choose an `action`, ask the micro:bit how long the game has been running for with `running_time()`.

2. Store that value in a variable called `start_time`.

3. On the next line, create a variable called `end_time`, set it to `start_time` plus 10,000 milliseconds (10 seconds). *You can change this later if you want a longer game!*

## Task 4.2: Here we go again!

**Now let's add the loop that goes until the end_time!**

1. Go to the next line after you set the `end_time`.

2. Add a `while` loop with a condition that checks that the current `running_time()` is less than the `end_time`.

3. `Indent` all the code that is below this line (your `if` statements), so they are inside the `while` loop.

## Hint

Your `while` loop should have a structure similar to this example:

```
while raining == True:
    print ('Raindrops keep falling on my head')
```

## Task 4.3: Wait a second and then change the action

We already show the image for the first action we choose! Let's wait 500 milliseconds, then choose a new action.

1. Make a new line below your second `if` statement. It **should be indented** inside the `while` loop, **but not** inside the last `if` statement.

2. Tell the program to `sleep` for `500` milliseconds.

3. After the `sleep`, update the value of `action` by choosing a new one from the list of `actions` again.

### Hint

To update a variable, just assign something new to it! You can use the same code you used in `Task 2.3` to pick the first random move.

## ☑ CHECKPOINT ☑

**If you can tick all of these off you can go to Part 5:**

☐ Your game runs for 10 seconds

☐ Your game keeps choosing new random actions

☐ Your game updates to the correct picture for the new action

# Part 5: Button Presses



You can't have a Bop It game without Buttons!
Let's make the game wait for you to complete the action.
Get it right to get a smiley face and a new action!
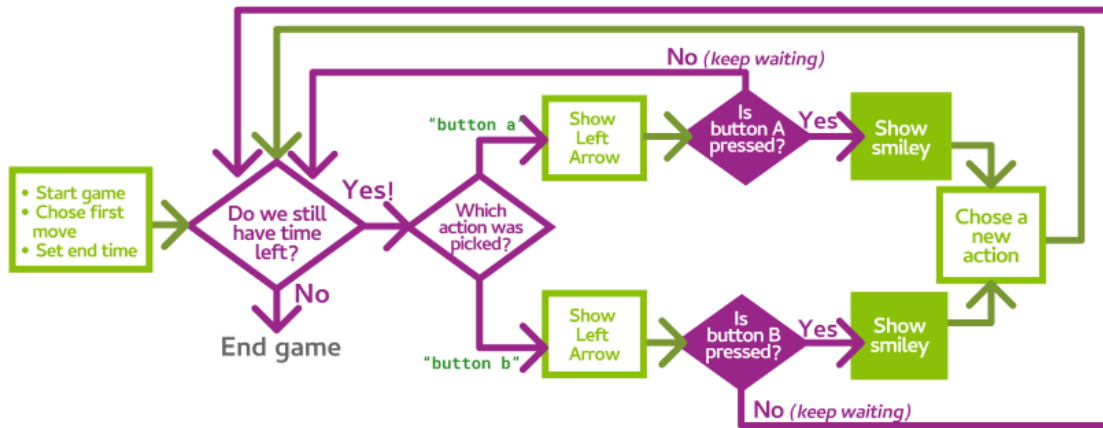
## Task 5.1: If A is pressed

**If the action is "press a", then we want to check whether button_a has been pressed.**

1. Create a new line of code after you display the arrow image for the **"press a"** action.

2. Create a new **if** statement to check whether **button_a.is_pressed()**.
   *Make sure this line is indented inside your existing **if** statement.*

3. Inside the new **if** statement, display a smiley face to celebrate!

### Hint

Nested if statements are tricky! It should look something like this

```
if weather == "sunny":
    if clouds == "white":
        display.show(Image.HEART)
```

### Hint - errors with is_pressed

Look at the end of this line of code, **notice the brackets at the end**:
**button_a.is_pressed()**

**Make sure you include the brackets!**
The brackets make it so we *call* the function and check if the button is pressed!

## Task 5.2: Else, when A is not pressed

**When button_a has not been pressed, we should continue the game.**

1. Add an `else` statement for if the button is not being pressed.

2. Inside that `else` statement, add `continue`.

## Hint

Your `if`-`else` statement should have a structure similar to this example:

```python
if raining == True:
  print ('oh no!')
else:
  print ('Yay!')
```

Don't forget that indentation is important!

## Task 5.3: Is B Pressed?

**Now it's button_b's turn!**

1. Inside the `if` statement that checks to see if the action is `button_b`, add an `if` statement that checks to see if `button_b.is_pressed()`.

2. Add an `else` to the `if` statement, that has a `continue`.

## ☑ CHECKPOINT ☑

**If you can tick all of these off you can go to Part 6:**

☐ When the action is **"press a"** and you press `button_a`, a smiley face is displayed.

☐ When the action is **"press b"** and you press `button_b`, a smiley face is displayed.

☐ Your game waits on the same move until you press the correct button.

# Part 6: Scoring



**A game needs a score!**
Let's make a score counter, add to it everytime we get it right and display the score at the end!

## Task 6.1: Let's get this scoring party started!

**Create a variable to keep track of the score.**

1. On a new line of code after you set `end_time` and before your `while` loop, create a new variable called `score`.

2. Set `score` to the value of `0`.

## Task 6.2: Get those points!

**Every time the correct move is made, add 1 to the score.**

1. Go to your `if` statement where you check if **button_a** is pressed.

2. On a new line, after where you show the smiley face, add 1 to the `score`.

3. Repeat for the other action.

### Hint - Keeping Count!

When we want to add to an existing variable it looks like this example:

```
num_apples = 5
num_apples = num_apples + 1
```

## Task 6.3: How did you do?

**Now we need to tell the player how well they did!**

1. Go to the very end of your code, after the `while` loop finishes.

2. Convert the final score to a string and then make it `scroll` across the display.

### Hint - String theory!

To scroll a number on the screen we need to convert it to a string.
We can use `str` to convert to a string inside our scroll, like this:

```
fave_num = 317
display.scroll(str(fave_num))
```

## ☑ CHECKPOINT ☑

**If you can tick all of these off you can go to the Extensions:**

☐ You have a score variable that is set to 0 at the start of the program.

☐ At the end of the game, the score scrolls across the display.

☐ You have made sure that the score counts to the right number.

# Extension Map

**Shake it off!**

The Micro:Bit can detect shaking!

Let's make Shake an action in our game!

**Don't shake and button! You'll tear your foil!** ❌

**Make your own buttons!**

By using the pins on the microbit we can attach up to 3 of our own button creations!

**Play that funky music!**

Games are better with sounds!

Use headphone and the music library to make your game more exciting!

**Create your own images!**

Are there images you want to use in your game that don't exist?

Learn how to create images yourself!!

**You are here!**

Your base
Bop It Game

**Wrong Button!** ❌

The real bop it game ends if you do the wrong action!

Let's make our game more challenging by detecting wrong actions!

**Oh No! Too Slow!**

The real bop it the game ends if you take too long to do an action!

Let's add a time limit to each move!

**Race your Friends**
(Multiplayer)

Sync up your game with your friends by starting your games at the same time with radio.
Race to 10 points.

**Race your Friends**
**Part 2**
**Game Master**
Create Game Master to trigger the start of your race. Wait till the first person is done and announce the winner.

**Race your Friends**
**Part 1**
**Player side**
Change your game to start on a radio signal. Keep playing till 10 points and radio that you're done!

**Co-op Bop!**
(Multiplayer)

Make a Bigger, Better Bop It, and play as a team!

Spread out your game by making buttons that talk to your game with radio!

**Co-op Bop!**
**Part 1**
**Radio Button**

Create a button that communicates via radio which you can join to a Co-Op bop it game.

**Co-op Bop!**
**Part 2**
**Game Master**
Change your game code to use radio buttons instead of regular buttons.

Add your friends' radio buttons to make a big team game!

**Bop Battle!**
(Multiplayer)

Create a multiplayer game where you battle for each point with your friends!

**Bop Battle!**
**Part 2**
**Game Master**

Create a Game Master that will send actions to all the players. Wait to see who completes them first and announce winners.

**Bop Battle!**
**Part 1**
**Player Side**

Create a player controller, it will receive actions by radio for the player to race to complete.