Girls' Programming Network

Bop It with micro:bits!

FOR TUTOR EYES ONLY

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

This workbook and related materials were created by tutors at:

Sydney and Canberra

Girls' Programming Network

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

Writers

Renee Noble
Courtney Ross
Alex McCulloch
Belinda Wong
Rowena Stewart
Kim Apted
Kassandra di Bona
Jennifer Henoch

Testers

Mikaela Goldstein
Michelle McPartland
Sheree Pudney

Intro to Micro:Bit

Part 0: Setting up

Task 0.1: micro:bits and pieces

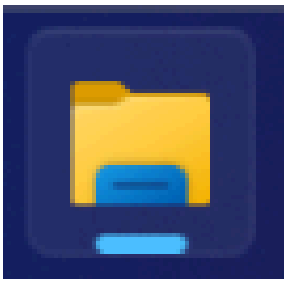
Let's set up the micro:bit for programming today! You should have:

- **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**

TUTOR TIPS

Students *must* use Google Chrome or Microsoft Edge to run code directly from python.microbit.org



If students do not have access to Chrome/Edge - they will need to download their code and move it onto their micro:bit manually.

To do this, click SAVE and then in File Manager drag the microbit .hex file from Downloads to the MICROBIT folder (which appears in File Manager after the micro:bit is plugged into the laptop)

Task 0.2: Micro playground

First we're going to play around with the displays on **python.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 (image.DUCK) and scroll your name instead
3. Test your code by clicking the arrow on the simulator.
4. Click the **'Send to micro:bit'** button to try this out. Follow any steps on the screen.
5. Try this out with other words and pictures.

Hint

Check out the cheat sheet for names of other images.
Remember to keep the code indented below the while loop!

☑ CHECKPOINT ☑

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

- ☐ You have connected your micro:bit to the computer
- ☐ You can display different pictures and words

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *

while True:
    display.show(Image.DUCK)
    sleep(1000)
    display.scroll("name here!")
```

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!

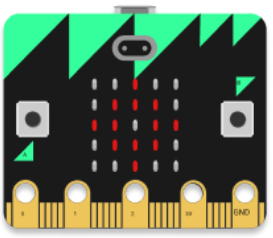
+ more

Once your base game works add cool extensions!
There's extensions for 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 **python.microbit.org** at the top of the page, edit your project name to be 'bop_it'.
2. At the top of the file add a new line (above the import line) and use a comment to write your name.

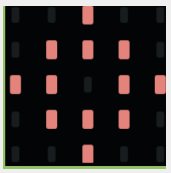
Hint

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()`**
4. Delete all the old code under your new **`display.clear()`** line
5. Test your code by clicking the arrow on the simulator
6. Click to 'Send to micro:bit' button to run it on the micro:bit

☑ 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 one second and then clears the display.

☐ You tried it on your real life micro:bit

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()
```

Random

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.

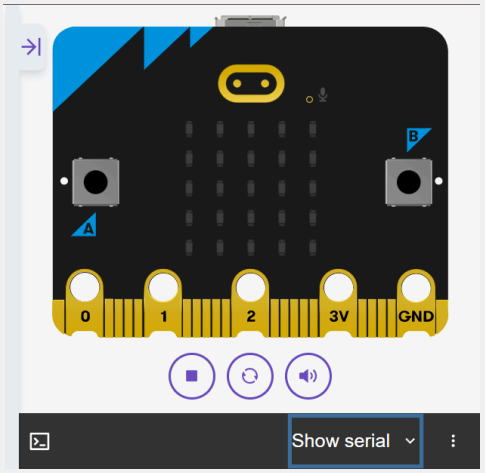
Underneath `from microbit import *`, add a new line of code that says `import random`

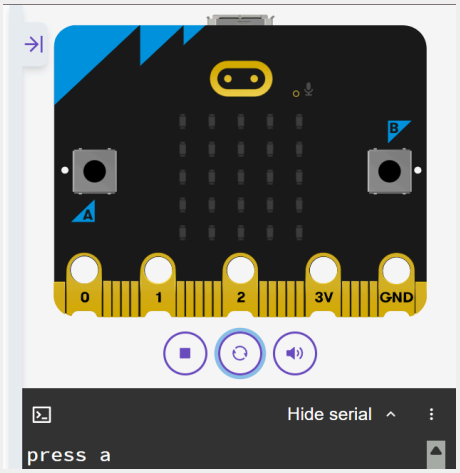
Task 2.3: Selecting the next action

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 `chosen_action`.
2. Choose a random action from the list of `actions` and assign it to `chosen_action`.
3. Then `print` the `chosen_action` so we can see what it is.

4. Run your code on the simulator
   It will print to the serial under the simulator micro:bit (click "Show serial" and scroll up to see it!)

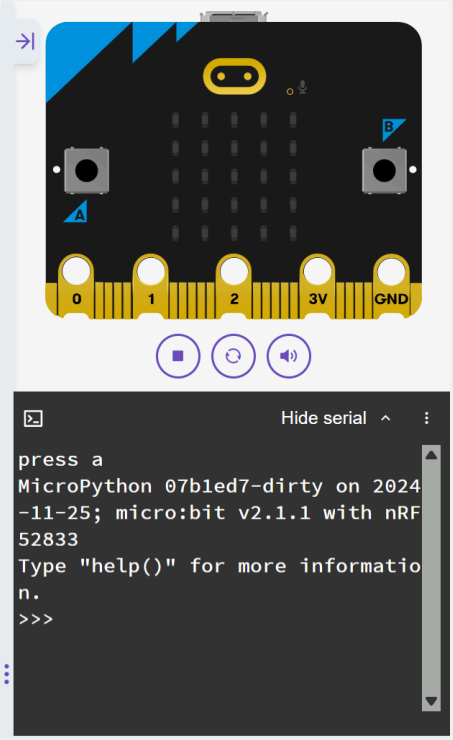




TUTOR NOTE: It is easy to overlook the print statement because of the prompt text following it. Scroll to the top of the serial output to see the output which will be either "press a" or "press b".

Students have to STOP their program running on the simulator (using the square STOP button under the simulator) so the run arrow will display again for the simulator.

There are two serials on the screen - one for the simulator (under the simulator) and one for when you run your code on the micro:bit (under the code). Make sure the student is looking at the correct serial.



Hint

Remember we can choose something randomly from a list like this:

```
fave_foods = ["pizza", "curry", "nutella", "omelette"]
dinner = random.choice(fave_foods)
```

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 on the simulator multiple times. See what action it prints out. You might have to scroll up to the top of the serial display to see the output.

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 **`chosen_action`** stored in a variable

☐ You have a print statement that **prints** out the **chosen_action**

TUTOR TIPS

The code should look like this (no bonuses):

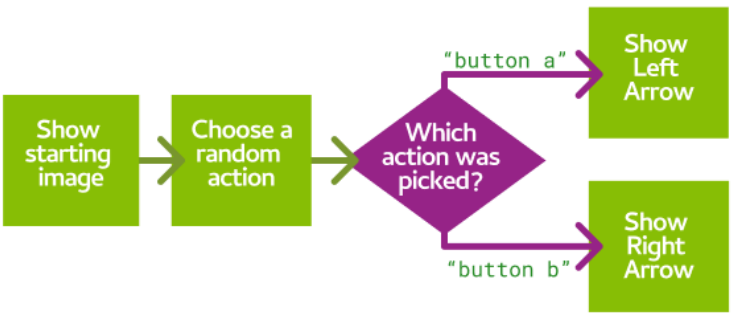
```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

actions = ["press a", "press b"]

chosen_action = random.choice(actions)

print(chosen_action)
```

If statements

Part 3: Light it up!

Let's tell the user which action we chose!

Display different images for each action!



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 these images for?

 **Image.ARROW_W**	 **Image.ARROW_E**
Action:	**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 `chosen_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.

Hint

Remember `if` statements have indentation. Here's an example about the weather:

```
if raining == True:
  display.scroll('Better find my umbrella!')
```

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 **serial** to see which `chosen_action` was selected.
2. Does the **micro:bit display** the correct picture for the randomly chosen action?

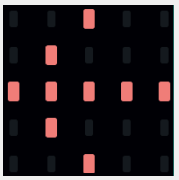
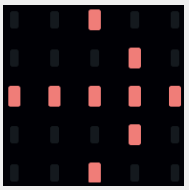
   Hint for tutors: There are two serials on the screen - one for the simulator (under the simulator) and one for when you run your code on the micro:bit (under the code). Make sure the student is looking at the correct serial.
3. Run your program multiple times to check both actions!

Once you feel confident that your code is working, you can delete or comment `print(chosen_action)`, as it is no longer needed.

☑ 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*:
https://girls-programming-network.github.io/content/Bop_It/Cheat%20Sheet%20-%20microbit-images.pdf

TUTOR TIPS

The code should look like this (no bonuses):

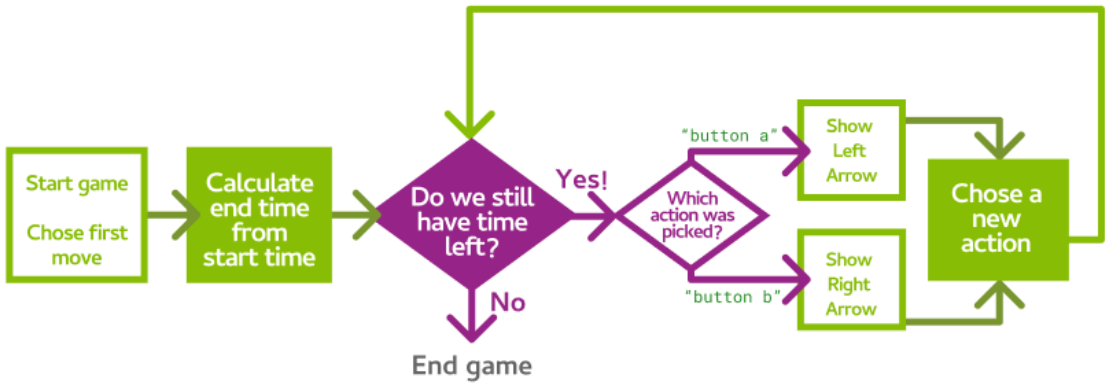
```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

actions = ["press a", "press b"]

chosen_action = random.choice(actions)

# If the action is press a
if chosen_action == "press a":
  display.show(Image.ARROW_W)

# If the action is press b
if chosen_action == "press b":
  display.show(Image.ARROW_E)
```

While Loops

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 **`chosen_action`**, ask the micro:bit how long the game has been running for with **`running_time()`** and store that value in a variable called **`start_time`**.
2. 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!*

Hint

`running_time()` is always counting the milliseconds since the program began. It helps you keep time in your Micro:Bit programs.

To figure out how long the Micro:Bit program has been running at a point in the program, you can store it in a variable:

```
time_since_start = running_time()
```

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 < maximum_allowed:
  display.scroll('It is going to get hotter')
```

You can indent existing lines using TAB on the keyboard.

You can indent multiple lines of code at the same time by selecting the lines and then hitting TAB.

Task 4.3: Wait a second and then change the action

We already showed the image for the first action we chose! 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 **`chosen_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

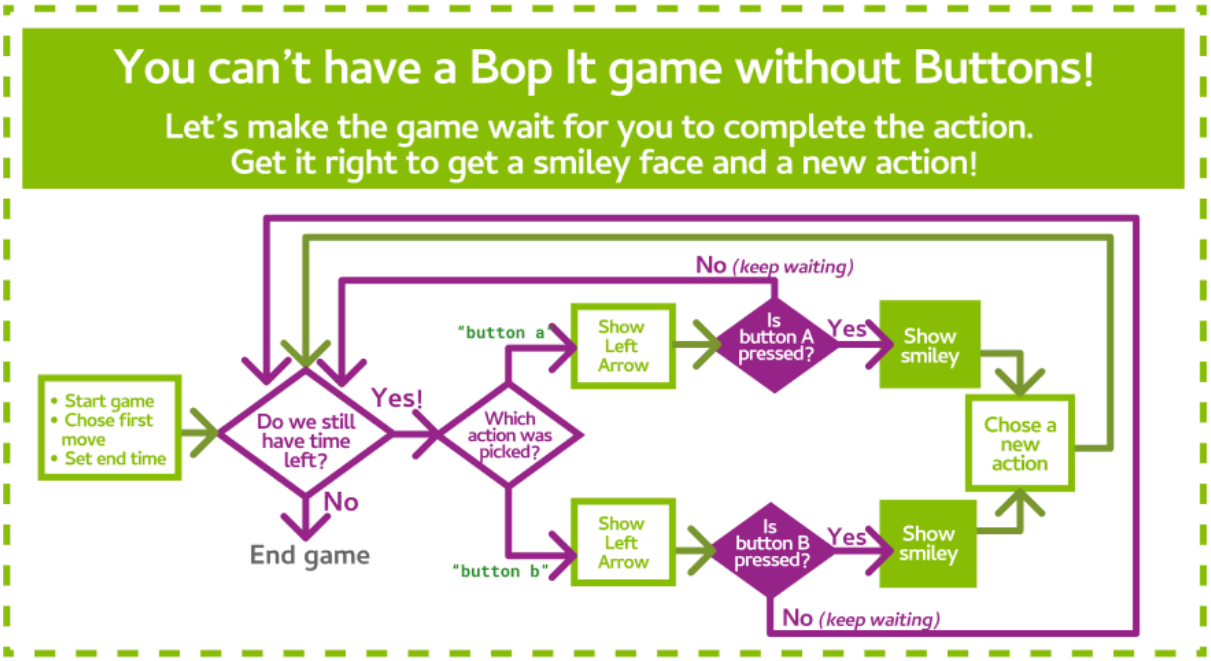
TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

actions = ["press a", "press b"]

chosen_action = random.choice(actions)

start_time = running_time()
end_time = running_time() + 10000

while running_time() < end_time:
    # If the action is press a
    if chosen_action == "press a":
      display.show(Image.ARROW_W)

    # If the action was press b
    if chosen_action == "press b":
      display.show(Image.ARROW_E)

    sleep(500)
    chosen_action = random.choice(actions)
```

Micro:Bit Buttons

Part 5: Button Presses



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:

```
if raining == True:
  display.scroll('Better find my umbrella!')
else:
  display.scroll('I don't need my umbrella!')
```

Don't forget that indentation is important!

Task 5.3: Button 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.

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

actions = ["press a", "press b"]
chosen_action = random.choice(actions)

start_time = running_time()
end_time = running_time() + 10000

while running_time() < end_time:
      # If the action is press a
      if chosen_action == "press a":
        display.show(Image.ARROW_W)
        if button_a.is_pressed():
          display.show(Image.HAPPY)
        else:
          continue

      # If the action was press b
      if chosen_action == "press b":
        display.show(Image.ARROW_E)
        if button_b.is_pressed():
          display.show(Image.HAPPY)
        else:
          continue

      sleep(500)
      chosen_action = random.choice(actions)
```

Part 6: Scoring

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** was 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. Got to the end of the program, 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

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

score = 0
actions = ["press a", "press b"]
chosen_action = random.choice(actions)

start_time = running_time()
end_time = running_time() + 10000

while running_time() < end_time:
      # If the action is press a
      if chosen_action == "press a":
        display.show(Image.ARROW_W)
        if button_a.is_pressed():
          display.show(Image.HAPPY)
          score = score + 1
        else:
          continue

      # If the action was press b
      if chosen_action == "press b":
        display.show(Image.ARROW_E)
        if button_b.is_pressed():
          display.show(Image.HAPPY)
          score = score + 1
        else:
          continue

      sleep(500)
      chosen_action = random.choice(actions)
display.scroll(str(score))
```