Girls' Programming Network

Tamagotchi 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, Melbourne and Perth

Girls' Programming Network

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

Writers

Sande Dalton
Paige Reeves
Sheree Pudney
Amanda Hogan
Isabella Hogan

Testers

Part 0: Setting up

Intro to Micro:Bit

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 **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 and scroll your name instead using `display.show(Image.DUCK), sleep(1000)` and `display.scroll("Your name")`
3. Click the **'Send to micro:bit'** button, then follow the steps on the screen.
4. Try this out with other words and pictures.

Hint: Cheat sheets

Don't forget you have cheat sheets on the web page 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 - Tamagotchi

We're going to make a Tamagotchi electronic pet!
By pressing buttons and moving the micro:bit you can interact with the pet to keep it happy and alive.

1. Start off by giving your pet a name!!
2. Show the pet's name, age, sleepiness, boredom and hunger.
3. Add the ability to play with, feed or send the pet to sleep.
4. Clean up the code!
5. Make the ability for the pet to die, but keep it alive!
6. Watch your pet grow up.

+ more

Once your base pet works add cool extensions!

Part 1: Welcome to Tamagotchi!

Task 1.1: Name your file!

Now you've been introduced to your micro:bit, let's start working on the project!

1. At the top of the page, edit your project name to be 'tamagotchi'.
2. At the top of your code, use a comment to write your name

Task 1.2: Welcome

Let's welcome our user and get things set up!

Before the while true loop, add the following:

1. Use `display.scroll` to say "Welcome to Tamagotchi"

2. Create some variables for our pet!
    a. `Name` (set this to whatever you want!)
    b. `Hunger`, `boredom` and `sleepiness` should all be set to 0

CHECKPOINT ☑

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

☐ Your Micro:bit scrolls the welcome message!

☐ You've tried it on your physical Micro:bit

Part 2: Seeing our pet!

Task 2.1: Shake it like a polaroid picture

If the Micro:Bit has been shaken, we want to display all of the current details about our pet. To do this let's first check if the board has been shaken.

1. Delete what's already in the while loop
2. Inside the `while True`, check to see if the accelerometer has detected the shake gesture
3. If it has, scroll what's currently in **`name`**

Hint: Using the Accelerometer

This is how you can test if the accelerometer (the inbuilt motion sensor) has detected a certain gesture.

```
if accelerometer.was_gesture("up"):
  display.scroll("Upwards!")
```

Task 2.2: What does our pet look like?

We now need to see what the pet actually looks like.

1. First, under where we are displaying the name, and still inside the if we want to display the image **`DIAMOND_SMALL`**
2. Then we need to **`sleep`** for 1 second

Hint: How to sleep?

Remember our sleep function works in milliseconds and one second is 1000 milliseconds so if we wanted to sleep for 5 seconds this is how we'd "**`sleep`**"

```
display.scroll("Z")
sleep(5000)
display.scroll("Z")
```

Task 2.3: How hungry, bored, and sleepy is our pet?

You should now scroll the values of each of our remaining variables

1. Still inside the if statement, scroll the text “Hunger:” and then the contents of our **`hunger`** variable then wait 50 milliseconds

2. Scroll the text “Boredom:” and then the contents of our **`boredom`** variable then wait 50 milliseconds

3. Scroll the text “Sleepy:” and then the contents of our **`sleepiness`** variable then wait 50 milliseconds

★ BONUS 2.4: Making your own photos!

We’re currently using an inbuilt image to represent our pet but that's a little boring

Create and display your own image for your Tamagotchi - ask us for paper grids to design with, or use existing images from the micro:bit Image Cheat Sheet: http://bit.ly/images-microbit

You can program in your own images pixel by pixel like this:

```
gpn_heart = Image("09090:33903:90009:03030:00900")
display.show(gpn_heart)
```

☑ CHECKPOINT ☑

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

- ☐ When your Micro:Bit is shaken, it displays the name
- ☐ It then displays an image for your pet
- ☐ It then scrolls to show the hunger, boredom, and sleepiness

Part 3: Interacting with our pet!

Task 3.1: Getting some Zzzs

We're going to flip our Micro:Bit over to send our Tamagotchi to bed.

1. Inside the while loop, make an if statement that checks if the accelerometer detects that the Micro:Bit is `"face down"`
2. Inside the if statement, reduce the **`sleepiness`** variable by 3 and show a sleepy face
3. We also only want to show this face for a couple seconds, so after you display the face, you'll need to use sleep, and then **`display.clear()`** to clear the screen

Task 3.2: Yummy yummy snacks!

We're going to make a new button to feed our Tamagotchi.

1. Inside our while loop, add a new if statement that checks if button **`A`** was pressed.
2. If **`A`** has been pressed, then decrease **`hunger`** by 3 and show a **`Happy`** face for a little bit, don't forget to clear the screen after!

Hint: Detecting buttons

Here's a reminder on how to test if a button **was** pressed

```
if button_a.was_pressed():
  display.scroll("AAA")
```

Task 3.3: Play time!

Now, we're going to play some games with our Tamagotchi.

1. Inside our while loop (again), add a new if statement that checks if button **`B`** was pressed.
2. If **`B`** has been pressed, then decrease **`boredom`** by 3 and show a silly face for a little bit, don't forget to clear the screen after!

☑ CHECKPOINT ☑

If you can tick all of these off, you can go to Part 4:

- ☐ If you turn the Micro:bit over, it decreases the Tamagotchi's sleepiness and shows a sleepy face
- ☐ If you press the Micro:Bit's **A** button it decreases the Tamagotchi's hunger and shows a smiley face
- ☐ If you press the Micro:Bit's **B** button it decreases Tamagotchi's boredom and shows a silly face.

Part 4: Cleaning up the code!

Task 4.1: Our first function!

Let's make our first function to clean up the display section of our code.

1. Make a function called **`show`** underneath where you're importing micro:bit
2. Copy all of your code from under the `if` statement that checks whether the micro:bit has been shaken into the show function
3. Now where your old code used to be, call the function.

Hint: Making a function

Remember this is how you make a function.

```
def my_function():
    # do some things here
```

Hint: Calling a function

Remember this is how you call a function to make it run.

```
def my_function():
    # do some things here

my_function()
```

Task 4.2: Sleeping, eating, and playing

Now we need to make a function to sleep

1. Make a function called sleeping and copy across all of your code for when the microbit is face down. You will need to set sleepiness to global at the top of the function

2. Where your old code was, call the function.

3. Now we need to repeat this process for when button_a and button_b are pressed, for functions called eating and playing

Hint: Global variables

Here's how to make a variable global in a function

```
def my_function():
    global my_variable
```

☑ CHECKPOINT ☑

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

- ☐ Your program has some functions now
- ☐ Your program works the same as it did
- ☐ You tried it on your real-life micro:bit

Part 5: Waiting and Dying

Task 5.1: Making a timer

Now we want to make a timer that starts a timer when we run the code, then resets every time 10 seconds has passed as we will mark that as one pet "year"

1. Directly under the functions, before any other code, make a variable called **`start`** and set it to the running time

2. Inside the while loop check if the current running time is more than 10,000 milliseconds greater than **`start`**. If it is, reset **`start`** to the current running time.

3. For now you might want to quickly display something to show that it's working

Task 5.2: Updating some attributes

Now we actually want to update all of our values after a "year" has passed.

1. Firstly, in the section where you're setting all your variables make a new one called **`dead`** this should start at `False`

2. You will need to make a function called **`wait`**. Then inside the function, write an if statement that checks whether dead is `True`. You should immediately return if it is.

3. You then need to replicate this if statement in all of your functions except show (this is so that the pet can't change if it's dead).

4. Now you will need to increase every value by 1 (That's **`hunger`**, **`boredom`**, and **`sleepiness`**) remember to make these variables global so that they can be edited in the functions.

5. Now, under where you increased all the values, you will now need another if statement that checks if **`hunger`**, **`boredom`**, or **`sleepiness`** are above 10. If even one is, you will need to set the pet to dead.

6. Then, in your while loop, above where you are updating **`start`** you should call the wait function so that everything updates

☑ CHECKPOINT ☑

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

- ☐ You now should have a timer that tracks a running time start
- ☐ It should also update all of the pet's attributes every 10 seconds
- ☐ It should be able to die if any of the values exceed 10

Part 6: We're growing Up, up, up!

Task 6.1: How old is our pet?

We're going to start storing the age of our pet so we can know how long we have kept it alive for!

1. In the section where you create all the variables create a new variable called **`age`** that starts at 0

2. Inside your wait function, add one to age in the section you're updating all the other attributes

Task 6.2: What does our Tamagotchi look like?

We need to tell our code what our Tamagotchi should look like as it grows up!
Create four new variables to hold images for our Tamagotchi:

1. Create a **`baby_image`** variable that holds Image.DIAMOND_SMALL

2. Create a **`child_image`** variable that holds Image.SNAKE

3. Create a **`adult_image`** variable that holds Image.BUTTERFLY

4. Create a **`dead_image`** variable that holds Image.GHOST

Task 6.3: Let's draw our tamagotchi

We need to draw our Tamagotchi as it grows up. To do this, we'll use a new function that draws it based on its current age.

1. Create a new function called **`get_pic`**

2. In this function, make an if statement that returns:
   - ***a.*** return **`dead_image`** if the Tamagotchi is dead
   - ***b.*** otherwise, if the Tamagotchi is alive and has **`age`** ≤ 3, return **`baby_image`**
   - ***c.*** otherwise, if the Tamagotchi is alive and has **`age`** ≤ 5, return **`child_image`**
   - ***d.*** otherwise, return **`adult_image`**

3. Then inside our show function, use this function to show the image `get_pic` returns

☑ CHECKPOINT ☑

If you can tick all of these off, you can go to Part 7:

- ☐ Your program has variables to store the baby, child, adult, and spirit micro:bit images you want to display.
- ☐ Your program has a get_pic() function.
- ☐ Your program displays your chosen baby image when the pet age is up to 3.
- ☐ Your program displays your chosen child image when the pet age is greater than 3 and up to 5.
- ☐ Your program displays your chosen adult image when the pet age is greater than 5.
- ☐ Your program displays your chosen spirit image when the pet is dead.
- ☐ You tried it on your real-life micro:bit

Extension 7: Talking Tamagotchi

Task 7.1: Feeling lonely

In this extension, we're going to be teaching your pets to say hello to each other.

Your micro:bits have an in-built **radio**, which allows them to both *send* messages to other micro:bits and *receive* messages from other micro:bits. Let's try it out!

1. Add a new state variable called **`loneliness`** and set its initial value to be 0.

From now on, in order to keep your tamagotchi alive, you'll need to manage its **`hunger`**, **`boredom`**, **`sleepiness`** *and* **`loneliness`** levels.

Task 7.2: Updating loneliness

Just like with hunger, your pet will get lonely if no one talks to them, and eventually, they can die from loneliness! Let's update our code to maintain the loneliness state.

1. Inside the **`wait`** function, where you increment your **`hunger`**, **`boredom`** and **`sleepiness`** levels, also increment **`loneliness`**.
2. Find where you've written an **`if`** statement setting your **`dead`** variable to **`true`**. Extend the condition so that your pet will also die if its **`loneliness`** level exceeds the threshold.
3. Finally, extend your **`show`** function to scroll **`hunger`**, **`boredom`**, **`sleepiness`** *and* **`loneliness`**.

Hint: Global variables

Don't forget to declare **`loneliness`** as **`global`**!

Task 7.3: Import radio

In order for your micro:bit to talk to other micro:bits, you'll need to import and use the **`radio`** module, just like you did with the **`microbit`** module.

1. At the top of your code, import **`radio`**.

2. Ensure the radio is switched on by adding the line **`radio.on()`** right after the code where you defined the **`start`** variable.

Hint: Importing radio

You can import the radio module by adding this line to the top of your code:

```
import radio
```

Task 7.4: Sending messages

Let's add a function for sending messages to other micro:bits.

1. Define a new function named `talk` with no parameters.

2. Inside the body of your function, you can use the `radio.send(message)` method to broadcast a message to any nearby micro:bits. Send a message introducing your pet, for example, `"Hello, my name is " + name`.

3. Add a line to display the message on your micro:bit, so that you can see what message you're sending out.

Make sure that your message ends with your pet's name. This will let other pets know who they should send their reply to!

Hint: Broadcasting messages

Here's an example of how we could broadcast the message "Hello world!"

```
display.scroll("Hello world!")
radio.send("Hello world!")
```

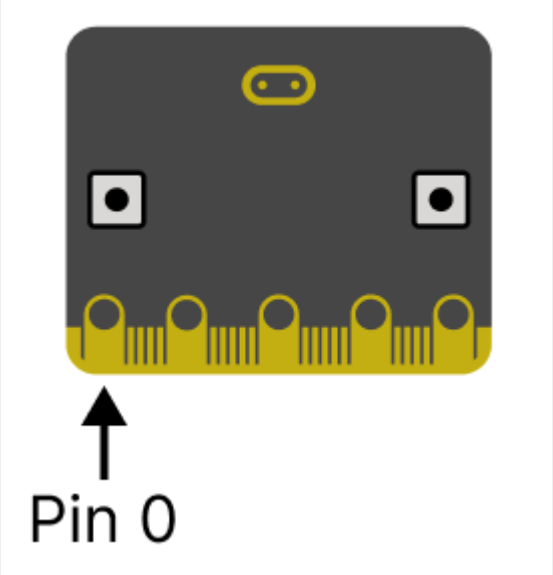
Task 7.5: Calling your function

Next, you can program your pet to broadcast its message whenever you touch **pin 0**. You can check if pin 0 has been touched by using the expression `pin0.is_touched()`.

1. Add a new `if` statement to your game loop to check if pin 0 was touched - if it was, call your function to say hello to other pets! This will cause some minor issues so you should put a delay of 1 second before moving on so it doesn't fire multiple times.

2. **Test it out**: load up your code onto your micro:bit, then touch pin 0 - you should see your message scrolling across the screen!

Hint: Micro:bit pins



Just like buttons A and B, your micro:bit also has several ***pins*** which you can use to interact with your pet. For this challenge, we'll be using pin 0, which is at the bottom left of your micro:bit.

Task 7.6: Being social

Sending out messages to other micro:bits is great, but not very helpful when we don't know how to receive messages yet! Our next challenge is to receive messages coming from other pets.
Let's start by creating a function for processing a message that you received from another pet.

1. Define a function called **`be_social`**, taking one parameter, **`message`**. That parameter will be a string representing a message you've received.

2. Remember that introduction messages end with a name - let's **`split`** the message into words and grab the last one, then we'll be able to tell who sent the message!
   In Python, we can use the **`split`** method to separate a sentence into words, then we can use **list indexing** to select the last word.

3. In the body of your **`be_social`** function, create a variable called **`sender_name`** and set it to the value of the *last word in the message.*

Hint: Splitting strings

Here's an example of how we can split the message **`"I love GPN"`** into words, and find the last word.

```
>>> message = "I love GPN"
>>> message.split()
["I", "love", "GPN"]
>>> message.split()[-1]
"GPN"
```

Task 7.7: Right back at you

Now that we know who sent the message, we can **`display`** our response.

1. Add a new line to your function that will scroll the words **`"Hi " + sender_name`** on your micro:bit's screen.

Task 7.8: Receiving messages

As our game is running, we'll need to regularly check our radio to see if anyone has sent us a message. This is just like how we needed to check if certain buttons were pressed in our `while True` statement. To check on our radio, we'll use the **`radio.receive()`** method.

1. Inside your `while True` statement, create a variable named **`message`** and set it to the return value of **`radio.receive()`**.

Task 7.9: Do you receive?

Let's take a look at how the method **`radio.receive`** works!
When you call **`radio.receive`**, there are two possible situations:

(1) Someone nearby has sent out a message. In this case, your micro:bit receives the message and **`radio.receive`** returns a string containing that message.

(2) No messages have been sent. In this case, your micro:bit does not receive any messages and **`radio.receive`** returns `None`.

`None` is a special value in Python which represents the *absence of data* - this is the micro:bit's way of telling us that there were no messages sent by anyone at all.

We only want to call our **`be_social`** function if we've actually received a message. That means we need an `if` statement to determine if the value of **`message`** is `None` or not. To check if the message is `None`, you'll need to use special syntax in your `if` statement:

```
if message is not None:
  # your code goes here
```

1. If the **`message`** received by your pet is not `None`, call the **`be_social`** function with **`message`** as your argument.

Hint: Displaying text

What's the difference between `is not` and `!=`?
Because `None` is a special value representing an absence of data, we need to use different syntax for comparisons. Don't forget to use `is` or `is not` when checking for `None`-ness, and `==` or `!=` when checking for real values!

Task 7.10: Testing it out

You're now ready to send and receive messages with other pets!

1. Check if a friend nearby has finished this section. If they have, load up your code onto your micro:bits, hold them close together, and have one of you touch **pin 0**. The *other* micro:bit should then scroll the message **"Hi, <name>"**

If you're the first person to finish this section, ask a tutor to help you out with testing your code.

☑ CHECKPOINT ☑

If you can tick all of these off, you can go to Extension 8:

- ☐ You can send messages to other micro:bits by touching pin 0
- ☐ You can receive messages sent from other micro:bits

Extension 8: Talking TO Tamagotchi

Task 8.1: Hearing noise

What if your pet has no other pets to talk to? It will die of loneliness! Wouldn't it be nice if **you** could talk to your pet and make it less lonely?

Micro:bits have in-built microphones which you can use to talk to your pet. After completing this challenge, you'll be able to talk to your pet and keep it happy, even when there are no other pets nearby.

Let's start by defining a **function** to call when the pet hears someone speaking into the microphone.

1. Define a function named **`hear_noise`**. This function won't need any parameters.
2. In the body of your function, write some code to make the text "Hello, human!" scroll across the screen.

Task 8.2: Feeling less lonely

When the pet hears you talking to them, they should be less lonely!

1. Add another line to the **`hear_noise`** function to *decrement* (decrease by 1) **`loneliness`**.

Hint: global variables

Don't forget to declare **`loneliness`** as `global`!

Task 8.3: Talking to your pet

Now, we just need to call the **`hear_noise`** function whenever the microphone is spoken into. To do that, we can use the **`microphone.current_event()`** method.

```
if microphone.current_event() == SoundEvent.LOUD:
  # call your function here
```

1. Add an `if` statement to your `while True` loop, underneath where you're checking the radio, to check if the microphone's current event is **`SoundEvent.LOUD`**.

2. Now, you can speak to your pet and make them less lonely! Try speaking into your microbit's microphone and check to see if it responds "Hello, human!"

☑ CHECKPOINT ☑

If you can tick all of these off, you're done!

When you talk to your micro:bit...

- ☐ Its loneliness decreases
- ☐ It displays a message in response