EdPy worksheets
Student worksheets and activity sheets

Lesson 6: Worksheet 6.1 – Flash the LED in response to a clap

In this activity, you need to write a program using Edison's clap-detecting sensor to get the robot to flash one LED light whenever it detects a clap.

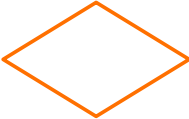The first thing to do is to plan out the program.

Flowcharts in programming

Professional programmers usually plan out their program before they start writing their code. Using flowchart diagrams is a way that programmers can organise and plan out their programs.

The idea of a flowchart is to graphically summarise what happens in the code without needing to go into all of the detail. Flowcharts allow a programmer to visualise and communicate how the 'flow' of the program will work.

In a flowchart, a program is represented using different shapes and arrows. Each shape represents a different element in the program, and the arrows show how the elements work together.

There are five main symbols used in flowcharts:

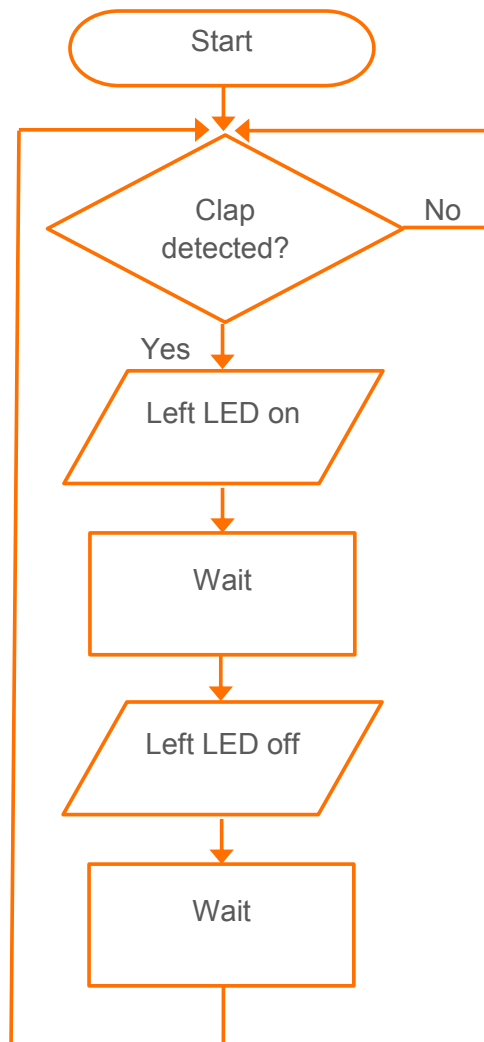| Symbol	Name	Function
⬭	Terminator (start/end)	An oval represents a start or end point
→	Arrow	A line which acts as a connector, showing the relationships between representative shapes
▱	Input/output	A parallelogram represents an input or an output
▭	Process	A rectangle represents a process or action
◇	Decision	A diamond represents a decision

More complicated flowcharts may also use additional shapes with different meanings.

When making a flowchart to plan a program, words are often added inside the shapes or next to the arrows. These words are short summaries of the process or decision.

Let's look at an example flowchart summarising the program we want to make.

Here is a flowchart for a program that will tell your Edison robot to wait for a clap, then flash the left LED:

```
                    ┌─────────────────┐
                    │     Start       │
                    └─────────────────┘
                            │
                            ▼
                        ◇ Clap
                          detected?  ──── No ──┐
                            │                  │
                           Yes                 │
                            ▼                  │
                    ╱ Left LED on ╱            │
                            │                  │
                            ▼                  │
                    ┌─────────────────┐        │
                    │     Wait        │        │
                    └─────────────────┘        │
                            │                  │
                            ▼                  │
                    ╱ Left LED off ╱           │
                            │                  │
                            ▼                  │
                    ┌─────────────────┐        │
                    │     Wait        │        │
                    └─────────────────┘        │
```

This program will use Edison's sound-detecting sensor to determine whether or not a clap has occurred. The result determines how the code flows next.

When you look at this flowchart, you may notice that it doesn't have an 'end' terminator. That is because this program uses a 'while' loop set up in a way to make the program continue indefinitely.

Making an infinite loop
Sometimes you may want to write a program that doesn't have an end, but loops forever. In programming, this is often referred to as an infinite loop.

You can use an infinite loop to make the program planned out in the flowchart.

Look at the following program:

```
1
2    #-------------Setup----------------
3
4    import Ed
5
6    Ed.EdisonVersion = Ed.V2
7
8    Ed.DistanceUnits = Ed.CM
9    Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below-----------
12   while True:
13       Ed.ReadClapSensor()
14       while Ed.ReadClapSensor() != Ed.CLAP_DETECTED:
15           pass
16       Ed.LeftLed(Ed.ON)
17       Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
18       Ed.LeftLed(Ed.OFF)
19       Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
20
```

This program is represented by our flowchart and includes an infinite loop.

Look at line 12 of the program, which uses a 'while' loop.

'While' loops always need a condition. The loop will repeat any indented code while that condition resolves to 'true'.

If we want the 'while' loop to repeat infinitely, instead of giving a condition the program must evaluate, we replace the condition with 'True'.

'True' always resolves to 'true'. By setting the condition to 'True', we have hardcoded the condition of our 'while' loop to be 'true'.

In programming, hardcoding is where you force something to be a specific way by explicitly typing it out.

By using 'True' as the condition for the 'while' loop in our program, the condition of the loop can never be false and will repeat infinitely.

Your turn:

Write the above code to program your Edison robot to make the left LED flash when you clap. Download it and test to see how it works.

1. What is the furthest distance away from your Edison that you can be and still have the robot sense when you clap?

   _____

2. What is the purpose of having the TimeWait() function calls in the code? What would happen if we didn't have them? *Hint:* Try running the program without the TimeWait() function calls.

_____

_____

_____

Lesson 6: Worksheet 6.3 – Design your own function

In this activity, you will design your own function and use it to write a program for Edison.

What, exactly, are functions?

Now that you have been programming for a while with Edison and EdPy, you have used a range of different functions from the EdPy library.

As you know, a function is a piece of code that performs a particular role or job in the program depending on which input parameters are used.

But you might not have realised that all of the functions you have used so far have actually executed multiple lines of code when run by the program.

That's because a function is a block of organised, reusable code that is used to perform a single, related action.

Functions are very helpful because they allow us to program is a more modular way, using the same block of code at various points throughout the program. To get your program to run all of those lines of code you only need to type one line: calling that function.
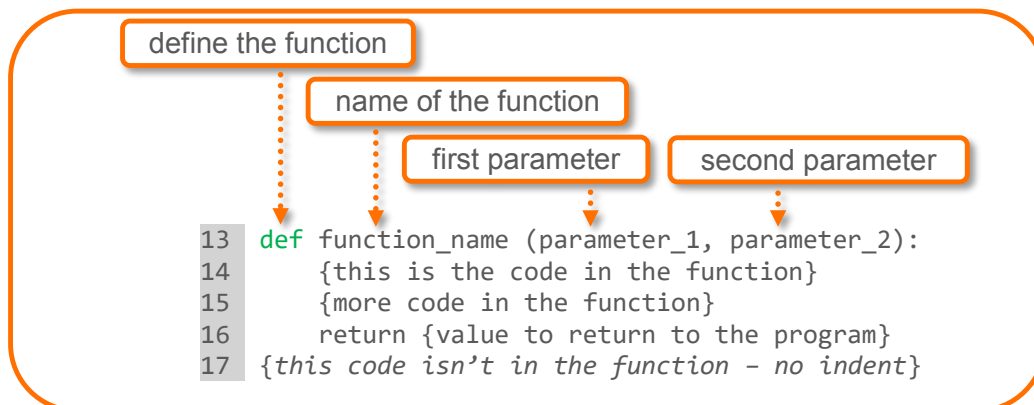
Functions make reusing code easier and more efficient when we program.

> Watch a short video by code.org explaining why functions are so useful in programming:
> https://youtu.be/8T5acEwfJbw

Making your own function

So far, every time you have used a function in EdPy you have called that function from the Ed library. You can also make your own functions.

In Python, functions look like this:

define the function

name of the function

first parameter

second parameter

```
13  def function_name (parameter_1, parameter_2):
14      {this is the code in the function}
15      {more code in the function}
16      return {value to return to the program}
17  {this code isn't in the function – no indent}
```

Anything indented is a part of the function. Anything not indented is not a part of the function, but the next line of code in the program.

It's important to note that the functions you create are no different from the functions you've used from the Ed library so far.

When you call your function (with or without parameters), the program jumps from the call to the code in the function (the indented code). The program then runs this code before returning to the line where you made the call.

If you set your function to return a value, when the program returns to the line where you made the call, the function call gets replaced by the value that the function has returned.

This is important because the program will always resolve functions, then maths orders then expressions in this order.

Organising your code

The convention in EdPy is to write the definition of your functions at the end of your code after you have already used your function in your program.

This is so your code is nice and neat. By organising your code this way, all your functions, whether you are using your own or calling functions from a library, can be written in the main part of your program in a visually clean, organised way. Your function definitions can sit outside of this, further down at the bottom of the program.

Your turn:

Task 1: Practice defining a function

Look at the following program:

```
 1
 2   #-------------Setup----------------
 3
 4   import Ed
 5
 6   Ed.EdisonVersion = Ed.V2
 7
 8   Ed.DistanceUnits = Ed.CM
 9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below-----------
12   directionToMove=Ed.SPIN_LEFT
13   speedToMoveAt=Ed.SPEED_8
14   distanceToMove=360
15
16   #call the function
17   moveOnClap(directionToMove, speedToMoveAt, distanceToMove)
18
19
20   #user defined functions
21   def moveOnClap(direction, speed, distance):
22       Ed.ReadClapSensor()
23       while Ed.ReadClapSensor()==Ed.CLAP_NOT_DETECTED:
24           pass
25       Ed.Drive(direction, speed, distance)
26
27
```

Line 17 in the program is the function call. Lines 21 to 25 are defining the function. Remember, the convention is to define your function at the bottom of your program to keep everything neat and organised.

Write the program and download it to your Edison robot. Run the program to see how it works.

1. We could write a function to make the Edison robot drive in a square shape. Fill in the missing words in the function below to finish writing this function.

```
def driveInaSquare():

    for i in range(___):

        Ed.Drive(_____,_____,_____)

        Ed.Drive(_____,_____,_____)
```

2. Write the syntax for the code to call this function. In other words, what would you type in your program to call this function?

_____

Write a program that will drive Edison in multiple squares. You will need to define the 'driveInaSquare' function, and your program will need to call this function more than once. Download and run the program to see how it works.

3. Does the program do what you expected it to do? If not, describe what you expected and what the program actually did. Describe any issues you had getting your program to work.

_____

_____

_____

_____

_____