

Micro:Bits

Micro:Bits Passwords

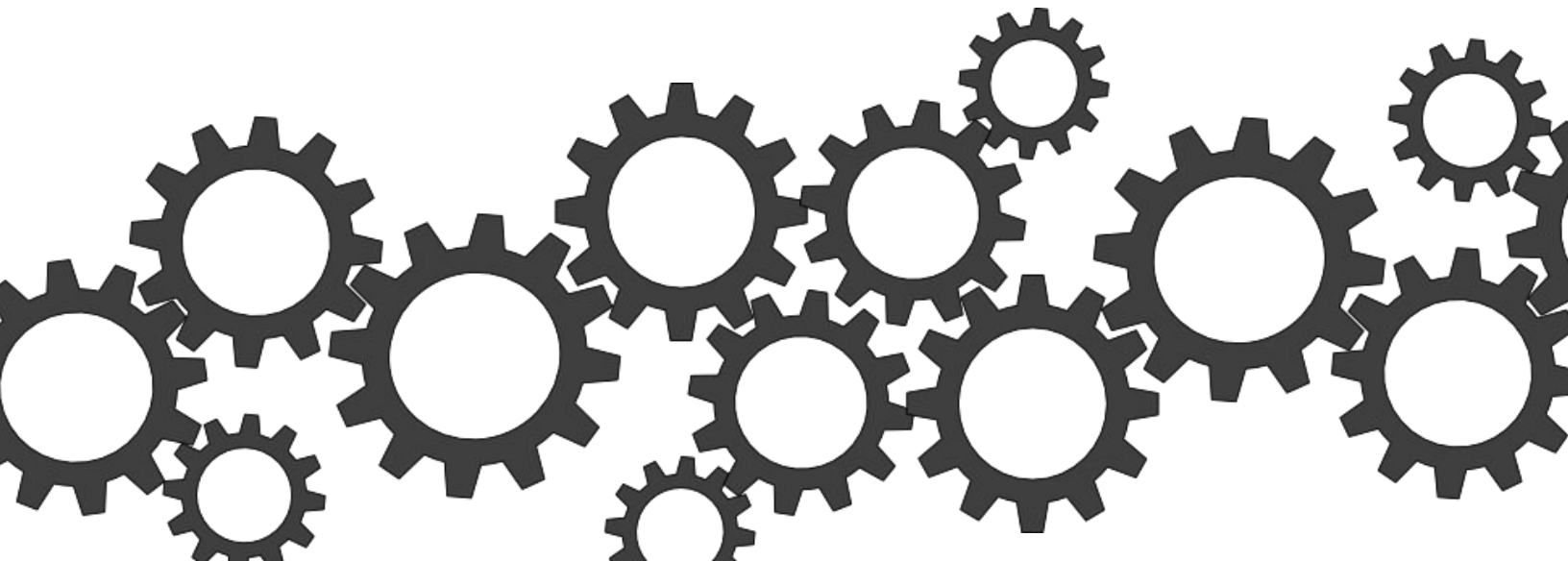
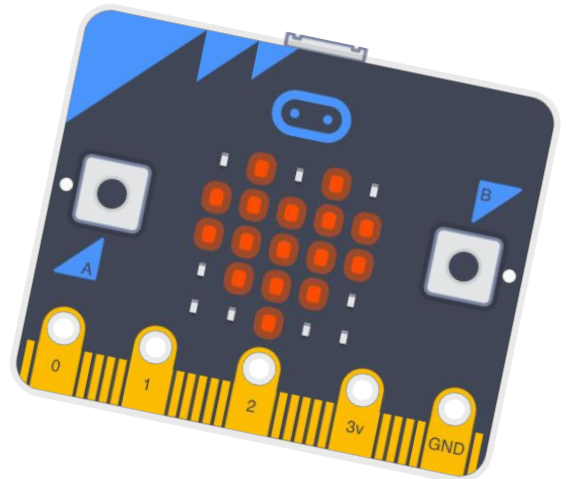
First, make a password generation program with Micro:bit Block Coding.
Then, make a version in microPython as well.

What is a Micro:Bit?

A Micro:bit is a miniature computer with a 5x5 LED display grid. We can use the MakeCode editor to code a program to execute on our Micro:bit. A program is simply a set of instructions for our Micro:bit to complete.

Materials

- Micro:bit kit
 - 1 Micro:bit
 - 1 Battery pack
 - 1 Micro USB cable
- Optional: 1 Dongle adapter if using Mac



Micro:Bits

Passwords

Procedure

- 1) Let's make a password generator. First, we will choose which characters we want to be in our password. Then we can add characters from that list to our password by pressing the A button. Then, we can press the B button to display the password we have generated. We'll add a feature where you can shake the micro:bit to erase the password and start again.
- 2) Let's start with the character options. Let's go to the array tab and create a list of characters called "char options" (character is frequently abbreviated as char in computer science). For the first part of the program, let's just add 1,2,3,a,b,c, so that there are 6 options total. Notice how big this block gets once we have six things in the list!




- 3) We will keep track of our password by adding characters to a new array one at a time. Duplicate the red block we just created and add it below. Click on the arrow next to char_options and create a new variable, called "password". Then, keep clicking the negative sign at the bottom of the array until it says "empty array".




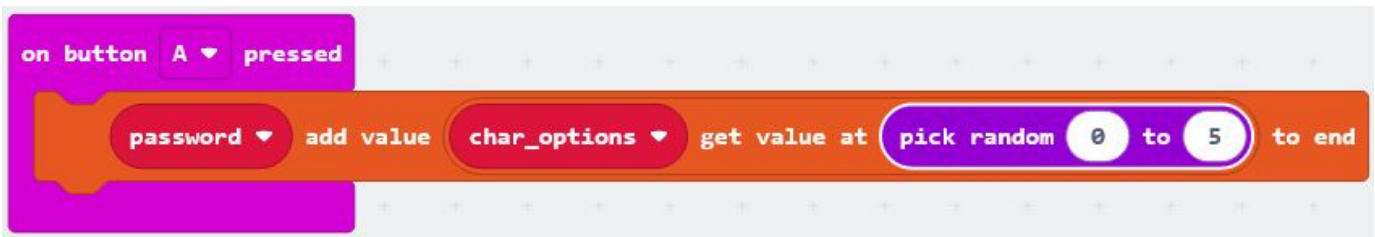
Micro:Bits Passwords



- 4) Drag out an “on button A pressed” block, then go to the Arrays tab and find this


block (under modify heading) . Also in the Arrays tab,

find this block . From the Math tab, bring out “pick random 0 to 10”. We want to get a random value from the character options, and add it to the end of our password. Can you see how to fit these three blocks together to do that? The final configuration looks like this:

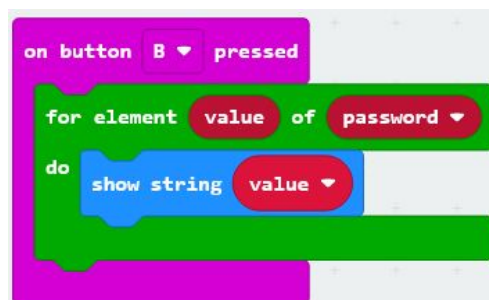


- 5) Notice that there are six characters in our char_options. The list counts the characters starting with 0, so they are listed off as 0, 1, 2, 3, 4, 5. That’s why we pick a number from 0 to 5, instead of from 1 to 6.

- 6) Now, let’s figure out how to display the password. Bring out an “on button B

pressed” frame, then go to the Loops tab and find this block . This loop is very useful anytime you need to do something with each element of a list. In this case, we are going to display each value in our password list. So, click the arrow next to ‘list’, and switch it to ‘password’.

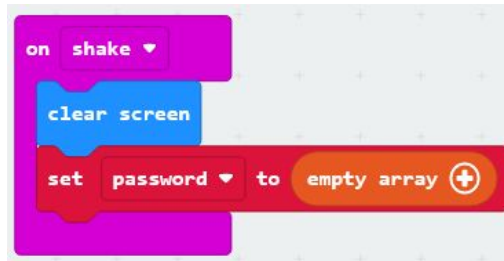
- 7) From the Basic tab, pull out a show string block for inside the loop. Then, click the rounded ‘value’ block and drag it into the show string block. Each time through the loops, the ‘value’ variable contains the next character of the password. The final result looks like this:



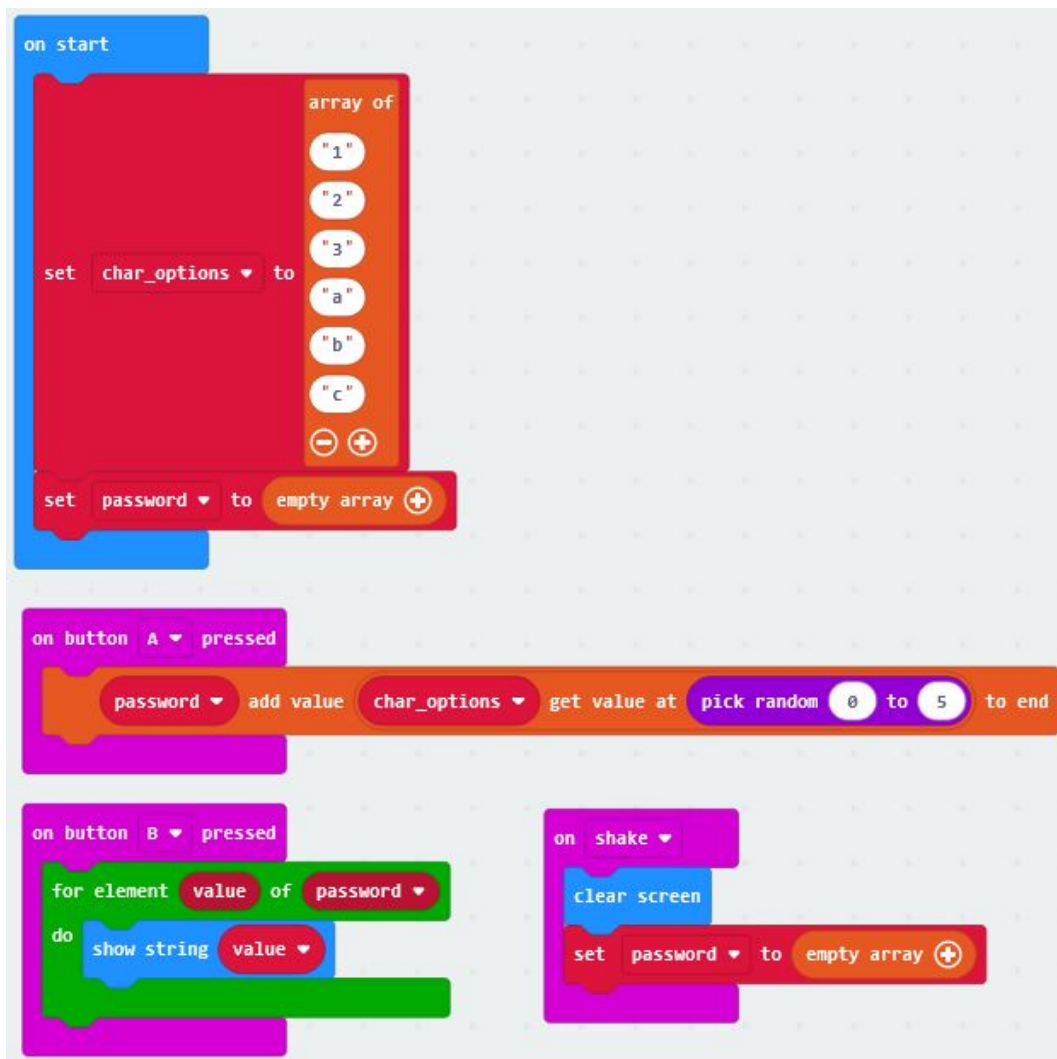
Micro:Bits

Passwords

- 8) Now, bring out an “on shake” frame from the Inputs tab. From the “on start” frame, duplicate the “set password to empty array” block, and put this into the on shake frame. Now, everytime you shake the micro:bit, the password is reset. Let’s also pull out a “clear screen” block from the Basic tab to put in the same frame, so that we can tell we have a fresh start.



- 9) Altogether, our final block coding version looks like this:



Micro:Bits

Passwords



- 10) Let's switch to Python now! We can try to learn how Python works with the Micro:bit by looking at how it translates block code into Python. Then, we can add more to our password generation program.



- 11) It can take some time to get used to Python, especially if Python is brand new to you! This is my first time using the Python language with Micro:bits, so I went through the program and added comments (green lines starting with #, an octothorpe).

```
1 # The def command is short for "define"
2 # def is used to define functions.
3 # Functions are blocks of code given
4 # a name, so that they can be easily
5 # ran by 'calling' their name.
6 def on_button_pressed_a():
7
8     # When you call "x.append(y)",
9     # y is added to the end of list x.
10    # We are adding char_options[randint(0, 5)]
11    # to the end of password.
12
13    # randint(0,5) picks a random number
14    # between 0 and 5.
15
16    # The number in the square brackets
17    # tells the program which character we want
18    # from char_options.
19    password.append(char_options[randint(0, 5)])
20
21 # This command tells the Micro:bit to
22 # call the "on_button_pressed_a" function
23 # everytime the A button is pressed.
24 input.on_button_pressed(Button.A, on_button_pressed_a)
```

Micro:Bits

Passwords



```
27 def on_button_pressed_b():
28     # This is a for loop in python.
29     # The first time through the loop,
30     # value holds the first character in password.
31
32     # Each time through the loop, value holds
33     # the next character in password.
34     # The loop ends after it uses the last
35     # character in password.
36     for value in password:
37         # Shows string "value" on the LCD screen
38         # Notice how the "show_string" function
39         # is from the "basic" tab.
40         basic.show_string("" + (value))
41 input.on_button_pressed(Button.B, on_button_pressed_b)
44 def on_gesture_shake():
45     # The global command tells the program that
46     # the 'password' variable we are talking about
47     # in this function is the same one that we
48     # defined in the "on start" frame.
49     global password
50     basic.clear_screen()
51     # Python uses square brackets to define lists
52     # so we can empty a list by setting it equal
53     # to a set of empty square brackets.
54     password = []
55 input.on_gesture(Gesture.SHAKE, on_gesture_shake)
57 # This is the code from the "on start" frame
58 # Seems weird to put it at the bottom of the program!
59 # The code above this actually runs first, but that code
60 # only defines what code will run when you press the
61 # buttons or shake the micro:bit.
62
63 # Tells the program that password and char_options
64 # will both be lists of strings.
65 password: List[str] = []
66 char_options: List[str] = []
67
68 # Initializes char_options with six characters
69 char_options = ["1", "2", "3", "a", "b", "c"]
70 # Initializes password as an empty list.
71 password = []
```

Passwords

- 12) That's a lot of new information to take in. Let's make some small changes by adding more characters to our char_options. Each value in the list needs to be separated by a comma. The values are strings, and we tell the program that they are strings by putting them in double quotes. I added some capital letters and some symbols to make the password harder to guess.

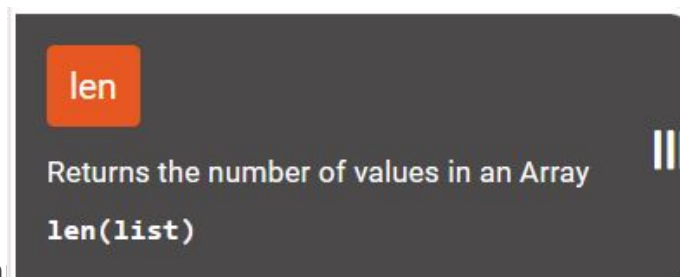
```
68 # Initializes char_options with characters
```

```
69 char_options = ["1", "2", "3", "a", "b", "c", "D", "E", "F", "&", "$"]
```

- 13) If you test your program now, you will realize that only the first six options are being chosen. Why don't our new characters ever show up? The problem is inside the randint command:

```
1 def on_button_pressed_a():
2     password.append(char_options[randint(0, 5)])
3 input.on_button_pressed(Button.A, on_button_pressed_a)
```

- 14) It only chooses integers between 0 and 5. We want it to choose between all of the possible indices for our char_options list. From our discussion before, you might have noticed that we need to choose an integer between 0 and one less than the number of items in our list. Go to the Array tab and click on 'len' to see this:



- 15) It is showing you a block that returns the number of values in an array called 'list'. You can even click and drag this block into your code, then change the text as needed. We need to replace the 5 in the randint command to be len(char_options)-1. Notice that suggestions show up when you start typing -- press tab to auto complete to the first option.

```
1 def on_button_pressed_a():
2     password.append(char_options[randint(0, len(char_options)-1)])
3 input.on_button_pressed(Button.A, on_button_pressed_a)
```

- 16) Now, we can add as many characters as we like to our list, without having to change any other code.

Micro:Bits

Passwords



Take Home

- 1) Make two lists, `char_options1` and `char_options2`. When you press A, append to your password from `char_options1`, and when you press A+B, append from `char_options2`.
- 2) First, let's split `char_options` into two lists, `char_options1` and `char_options2`. Notice there are two parts to initializing `char_options`: the type declaration in one line, then the actual initialization in the second line. The type declaration will be the same (except the name changed), but the initialization for `char_options2` will contain a different set of symbols.

```
20 char_options: List[str] = []
21 char_options = ["1", "2", "3", "a", "b", "c", "D", "E", "F", "&", "$"]
```

- 3) After splitting `char_options` up, the two lines above should become four lines, looking something like below. Keep in mind that you can have the two lists contain whatever symbols you like, as long as they are valid symbols for your password! I had numbers and lowercase letters in the first list, then capital letters and other symbols in the second list.

```
20 char_options1: List[str] = []
21 char_options2: List[str] = []
22 char_options1 = ["1", "2", "3", "a", "b", "c"]
23 char_options2 = ["!", "@", "#", "A", "B", "C"]
```

- 4) We will also have to go up to our function definition for `on_button_pressed_a` to change each instance of `char_options` to `char_options1`. Notice there are two places you need to make that change:

```
1 def on_button_pressed_a():
2     password.append(char_options1[randint(0, len(char_options1)-1)])
3     input.on_button_pressed(Button.A, on_button_pressed_a)
```

Micro:Bits

Passwords



Take Home

- 5) Finally, we need to add a function definition for “on_button_pressed_ab”, so that it is possible to add characters from list 2 to our password. This will be the same as the “on button pressed a” definition, except there will be two places where you change ‘char_options1’ to “char_options2”, two places where you need to change “a” to “ab”, and one place where you need to change “A” to “AB”. Can you find them all?

```
1 def on_button_pressed_a():
2     password.append(char_options1[randint(0, len(char_options1)-1)])
3     input.on_button_pressed(Button.A, on_button_pressed_a)
4
5 def on_button_pressed_ab():
6     password.append(char_options2[randint(0, len(char_options2)-1)])
7     input.on_button_pressed(Button.AB, on_button_pressed_ab)
```

- 6) There’s one last issue that you may have noticed at this point. If your password contains the same symbol twice in a row, you can only tell because the symbol is displayed for longer. Let’s make that clearer by adding a space before each character is displayed. That way, the scrolling screen tells us that we are looking at the next letter in the password. To do this, replace the empty quotes, (“”), in the display string command with quotes containing a single space (“ ”).

```
9 def on_button_pressed_b():
10     for value in password:
11         basic.show_string(" " + (value))
12     input.on_button_pressed(Button.B, on_button_pressed_b)
```

Micro:Bits Passwords



Final Program

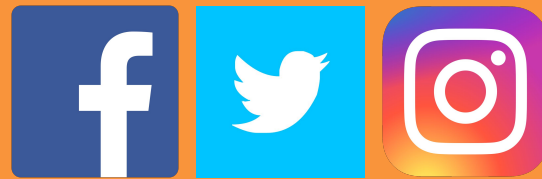
```
1 def on_button_pressed_a():
2     password.append(char_options1[randint(0, len(char_options1)-1)])
3 input.on_button_pressed(Button.A, on_button_pressed_a)
4
5 def on_button_pressed_ab():
6     password.append(char_options2[randint(0, len(char_options2)-1)])
7 input.on_button_pressed(Button.AB, on_button_pressed_ab)
8
9 def on_button_pressed_b():
10    for value in password:
11        basic.show_string(" " + (value))
12 input.on_button_pressed(Button.B, on_button_pressed_b)
13
14
15 def on_gesture_shake():
16     global password
17     basic.clear_screen()
18     password = []
19 input.on_gesture(Gesture.SHAKE, on_gesture_shake)
20
21 password: List[str] = []
22 char_options1: List[str] = []
23 char_options2: List[str] = []
24 char_options1 = ["1", "2", "3", "a", "b", "c"]
25 char_options2 = ["!", "@", "#", "A", "B", "C"]
26 password = []
```

What changes would you make to continue to improve this program?

#SVatHome

Want to share your
project or results with us?

Email or tag us
@ScienceVenture



Have a question?

Reach us at
svcamp@engr.uvic.ca