Jumpstart Python for micro:bit

Data Types

Data just means information your code works with.

String - *a sequence of characters, like words or sentences*

```
name = "Firia Labs" #or 'Firia Labs'
city = "Madison" #or 'Madison'
display.scroll("CodeSpace")
```

Convert an integer or float to a string with `str()`

```
display.scroll(str(7)) #Converts to "7"
```

Integer - *a whole number that can be pos, neg, or zero*

```
zip_code = 35758
num_trombones = 76
```

Convert a decimal or string to an integer with int()

```
int(7.9) #Returns 7
int("25") #Returns 25
```

Boolean - *a value that can be True or False; Zero values and empty strings are False*

```
bool("") #Returns False
bool("String") #Returns True

bool(0) #Returns False
bool(1) #Returns True
```

List - *a sequence of items you can access with an index*

```
colors = ["red", "green", "blue"]
colors [0] #Returns red
colors [1] #Returns green
colors [2] # "blue"
```

Count the length of a list with `len()`

```
len(colors) #Returns 3
```

Float - *a real number with a decimal point*

```
temperature = 98.6
pi = 3.141592
pi = round(pi,2) #Returns 3.14
```

Convert an integer to a float with `float()`

```
float(7) #Returns 7.0
```

Tuple - *a sequence of immutable objects, similar to lists. The difference between tuples and lists is that tuples cannot be changed.*

```
music.play(music.NYAN) #You can play the song, but you cannot change any individual notes.
```

Import Statements

Lets you use code from modules or libraries outside your source file.

*To provide access to the ENTIRE built-in micro:bit code. * is a wildcard and is shorthand for "everything"*

```
from microbit import *
```

To access part of a module, call only the PART you want

```
from random import randrange
random.randrange(10) #Returns random integer 0-9
```

Loops

Repeating code, subject to conditions you give.

***While loops:** the statement repeats the indented block of code while the condition is true.*

```
while loops < 30:
    loops = loops + 1

while True: #Forever loop, because
    True == True!
```

Variables

Memory space for storing things.

Must begin with a letter or _ , but may contain letters, numbers, and _ .

***Global variables**: variables defined outside of a function*

```
my_favorite_number = 73
num = 8
n = n + 1 or n += 1
```

***Local variables**: variables inside functions*

```
def spin_animation(num):
    delay = 50
    index = 0
    loops = 0
```

Functions

A chunk of code you can use by calling its name.

In other programming languages, they are also called procedures.

```
#Define a new function
def flash_smile():
    display.show(Image.HAPPY)
    sleep(500)
    display.clear()
    sleep(500)

#Call the function
while True:
    flash_smile()
```

Branching

Decision points in code.

Your code with take a different branch depending on values or conditions.

```
if condtion_A:
    #Do something amazing!
elif condition_B:
    #elif is short for "else if"
    #Do this only if condition B is
    true and condition A is false
else:
    #Finally, do this if A and B are
    false
```

Comparison Operators

Expressions let you compare two values. The result of a comparison is a True/False Boolean value.

```
> Greater than
< Less than
== Is equal to
!= Is not equal to
>= Is greater than or equal to
<= Is less than or equal to

value = 5 #5 is assigned to value
x > 10 is False
x < 10 is True
x == 10 is True
```

Editor Shortcuts

Keyboard hotkeys to write code faster. On PCs, use the control key (ctrl); on Macs, use the command (⌘) key

ctrl + x = cut; removes from the editor to be pasted later
ctrl + c = copy; stores text to be pasted later
ctrl + v = paste; inserts stored text
ctrl + z = undo; undo the last action
ctrl + f = find; search for text in your program
ctrl + / = comment; toggles '#' in front of line

FIRIA LABS

https://make.firialabs.com/

Inputs and Outputs

Buttons

Read input statements from Buttons A or B

```
button_a.was_pressed()
#Returns True if button A has been
pressed since the last call

button_a.is_pressed()
#Returns True if button A is
currently pressed

button_b.get_presses()
#Returns the number of times button B
has been pressed since the last call
```

Accelerometer

Measures the force of acceleration in the x-, y-, and z-axis

```
accelerometer.get_x()
accelerometer.get_y()
accelerometer.get_z()
#Returns a value from -2048 to +2047

accelerometer.current_gesture()
accelerometer.was_gesture()
```

Gestures can be:

```
3g, 6g, 8g, up, down, left, right,
face up, face down, freefall, shake
```

Light Sensor

Measures the ambient light and returns as an analog value

The micro:bit actually uses the LEDs of the display as a light sensor.

```
display.read_light_level() #Senses
ambient light. Returns an integer
between 0 (dark) and 255 (bright).

while True:
    light_level =
    display.read_light_level()
    display.scroll(str(light_level))
```

Magnetometer

Measures magnetic field strength and direction

Get the magnetic field strength around the device, or along one axis.

```
compass.get_field_strength()
compass.get_x()
compass.get_y()
compass.get_z()

compass.calibrate()#Start compass
calibration

compass.heading() #Returns the
compass heading 0-360, 0 as N
```

Music

Melodies that can be imported from a music library

```
import music
```

Defaults are `pin0, wait=True, loop=False, ticks=4, bpm=120.`

```
music.play(music, pin=microbit.pin0,
wait=True, loop=False)

music.stop(pin=microbit.pin0)

music.set_tempo(ticks=4, bpm=120)

music.pitch(frequency, duration=-1,
pin=microbit.pin0, wait=True) #Play
a sound for 'duration' in ms. -1
means continuous
```

Radio

Built in; can communicate with other micro:bits

```
import radio

radio.on()
radio.config(channel=N) #0-83

radio.send("message") #Send a
message string over the air

radio.receive() #Return a message
string if one has been received, or
an empty string if one has not
```

Display

A 5x5 LED display

Defaults are `wait=True, loop=False`. *If wait is True, this function will block until the animation is finished, otherwise the animation will happen in the background. If loop is True, the animation will repeat forever.*

```
display.show(value,delay=400,
  wait=True,loop=False, clear=False)
  #Display these images in sequence

display.scroll(value, delay=150,
wait=True, loop=False)
#Scrolls value horizontally

display.clear() #Set all LEDs to 0
(off)

display.off() #Turn off the display
allows you to re-use the GPIO pins
associated with the display

display.on() #Turns the display on
```

To make a custom image, create a string that looks like the display.

```
boat = Image("05050:"
             "05050:"
             "05050:"
             "99999:"
             "09990")

display.show(boat)


display.set_pixel(x, y, value) #Set
the brightness of the LED at column x
and row y to a value between 0 and 9.

display.get_pixel(x, y, value)
#Return the brightness of the LED at
column x and row y as an integer
between 0 (off) and 9 (bright).

fill(level) #Set the brightness
level
```

Pins

Input/Output connections

The pins are your board's way to communicate with external devices connected to it. There are 19 pins for your disposal, numbered 0-16 and 19-20. Pins 17 and 18 are not available.

Digital

Finite values - 0 and 1

Returns 1 if the pin is high and 0 if it's low.

```
value = pin.read_digital()
pin.write_digital(1)
```

Analog

Values with an infinite variation

The value may be either an integer or a floating point number. Typically 0-1023 or 0-255, etc.

```
pin.write_analog(400)
value = pin.read_analog()
```

Time

Controlling the pace of actions

```
sleep(1000) #Delay function 1000 ms
or 1 second
value = running_time() #ms since
last reboot
```