Intro to Python

Learn Python using the Adafruit Circuit Playground Express and CircuitPython

Instructor: Robert “Jack” Babb

2/23/19

What we are going to cover

- Introductions
- Python Basics
- Circuit Playground Express
- Installation of CircuitPython and Mu editor
- Some simple programs
- Sound to Light project

Questions

- What do you hope to get out of this class?
- Have you done any programming ?
  - Language: assembler,compiled,interpreted?
- Have you used a microcontroller?
  - Peripherals? DAC,ADC,SPI,I2C,GPIO,Timers
- Are you familiar with number systems?
  - Hex, Octal, Binary, Decimal

Advantages of Python

- Good scripting language
- Most popular scripting language (Java/c/c++ are more popular TIOBE Index for December 2018)
- Used in a wide array of applications: Web, Data intensive, Scientific, AI
- Easily adapts to multiple development styles: procedural, object oriented, functional
- Available on a wide variety of platforms
- Powerful collection and iteration abstractions
- Dynamic typing simplifies coding

Python Background

- Developed by Guido van Rossum in the early 1990s
- Named after Monty Python
- Freely available for download from http://www.python.org
- Many different versions available:
  - Full Python (Cpython) runs on Linux (Rpi), Windows, Mac, Android
  - Micropython designed for microcontrollers: https://micropython.org/
  - Adafruit Circuitpython, simplifed version of Micropython: https://github.com/adafruit/circuitpython

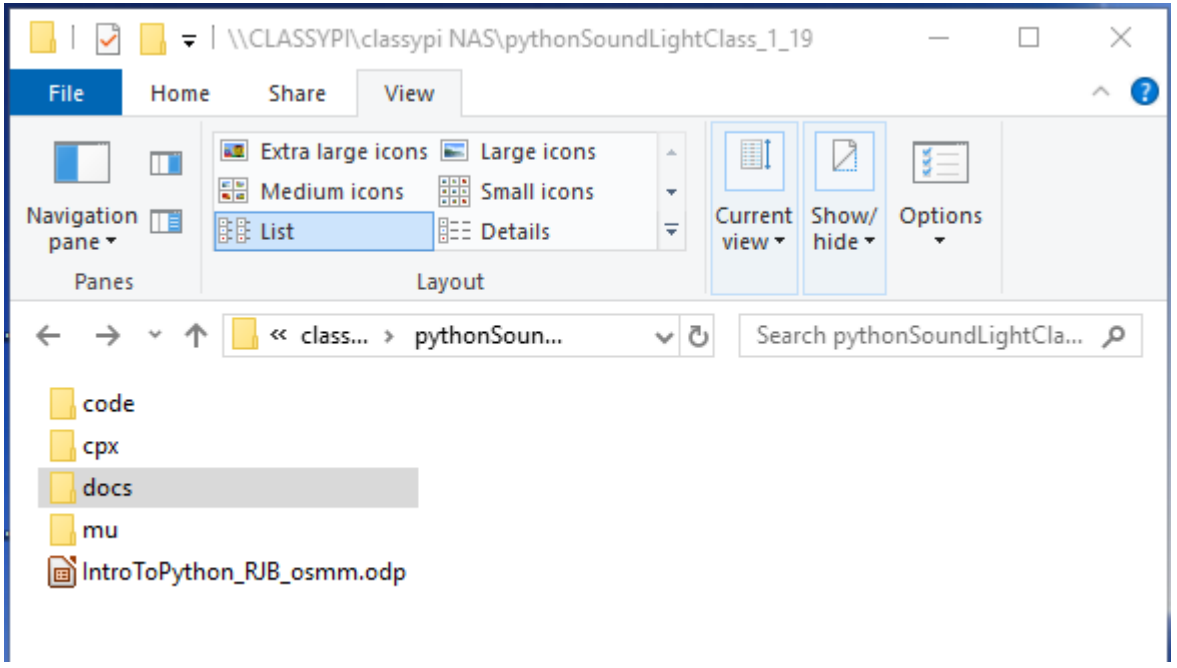
Meet the Adafruit Circuit Playground Express

- Open your CPX kits
- Pull out CPX and USB cable

Copy Class Files

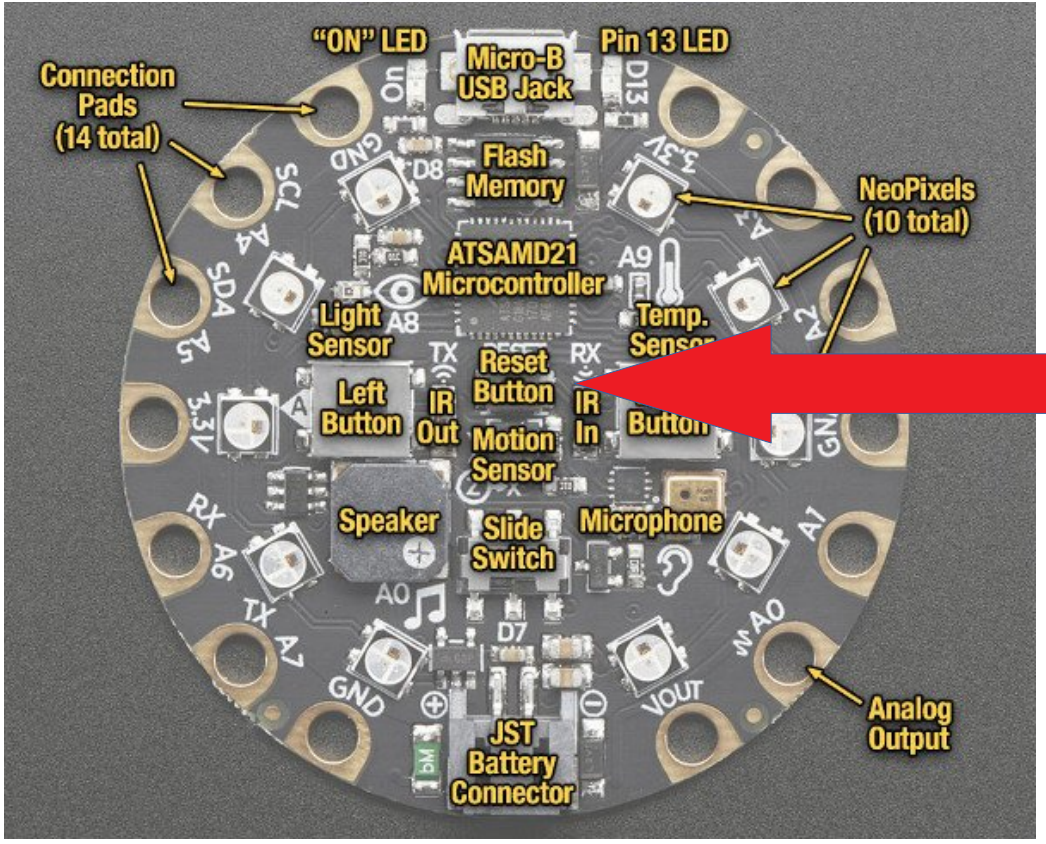
- Connect to Rpi_Hotspot WiFi
  - Password: 1234567890
- Copy files from: “\\CLASSYPI\classypi NAS” share
  - Zip file or directory tree: pythonSoundLightClass_1_19
- If you have trouble accessing the files, ask for the USB thumb drive

Class Files



- code: example programs
- cpx: Circuit Playground Express firmware
- Docs: references
- Mu: editor install
- IntroToPython…: class slides

Adafruit Circuit Playground Express



Install CircuitPython

- Plug your Circuit Playground Express into your computer
- Double-click the small Reset button in the middle of the CPX, you will see all of the LEDs turn green.
- You will see a new disk drive appear called CPLAYBOOT
- From the class CPX directory drag the adafruit-circuitpython-etc...uf2 file onto CPLAYBOOT
- The CPLAYBOOT drive will disappear and a new disk drive will appear called CIRCUITPY

Install Mu Editor

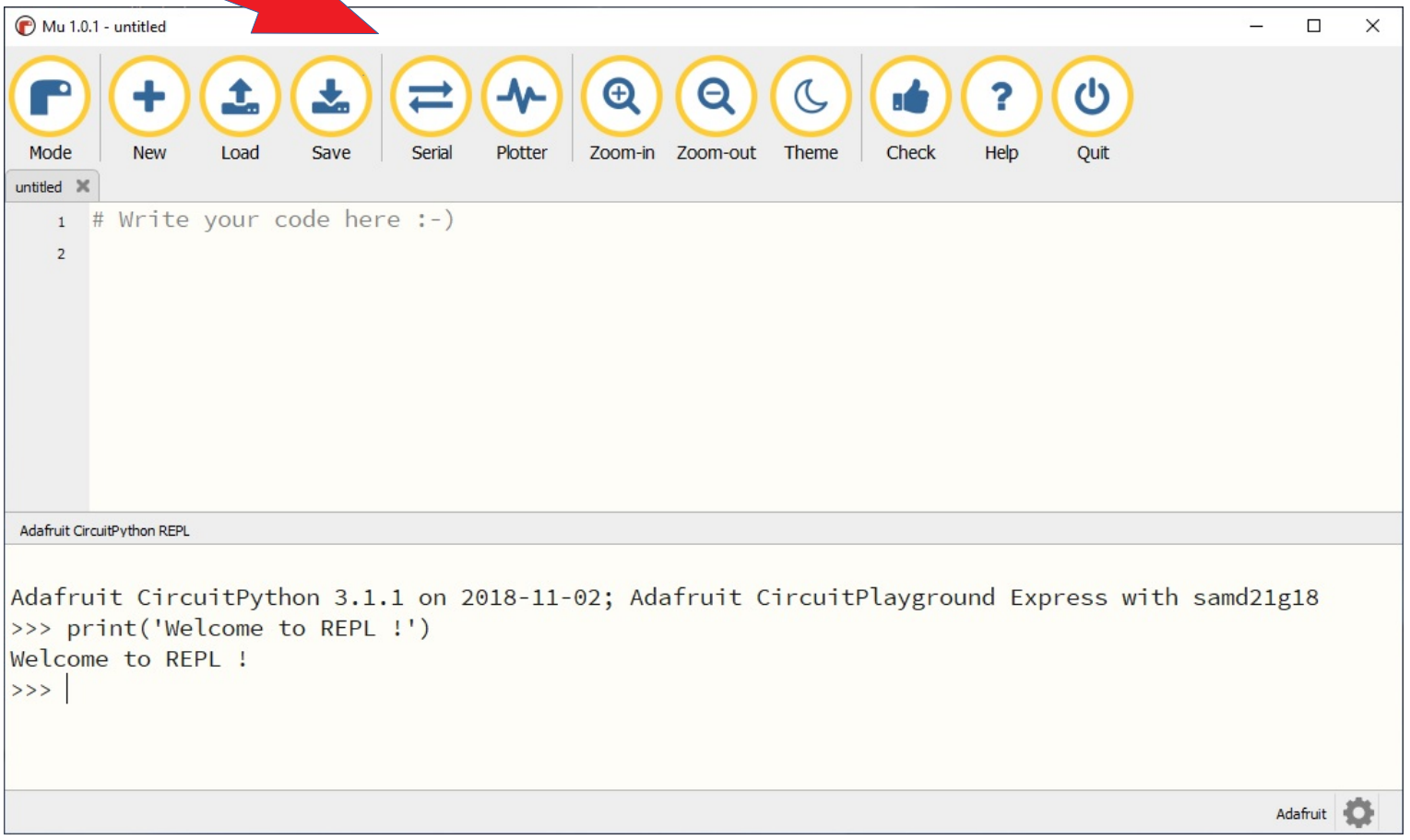
- Goto the class Mu directory
- Download the correct 32 bit or 64 bit installer for your operating system (probably: mu-editor_1.0.1_win64.exe)
- Double-click the installer to run it
- Mu attempts to auto-detect your board, make sure it shows up as a CIRCUITPY drive
- Run Mu
- select your 'mode' - please select Adafruit!

Python Playground: REPL

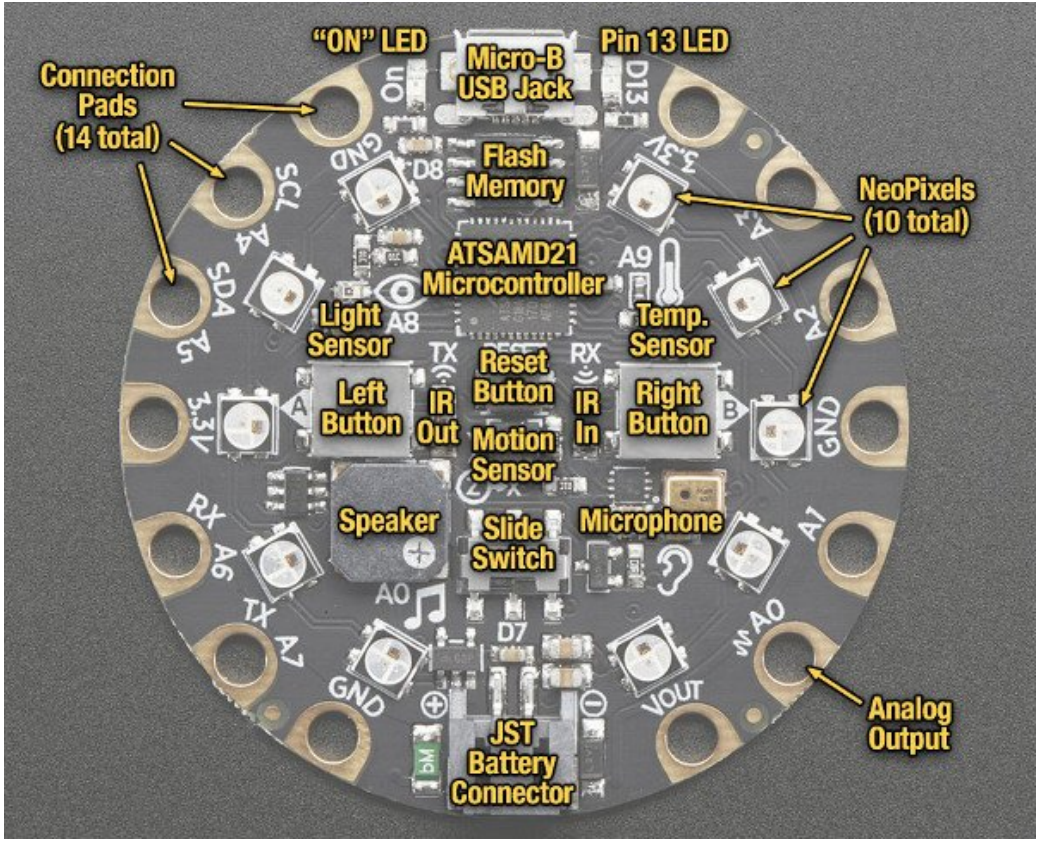
A REPL (say it, “REP-UL”) is an interactive way to talk to your computer in Python. To make this work, the computer does four things:

- **R**ead the user input (your Python commands).
- **E**valuate your code (to work out what you mean).
- **P**rint any results (so you can see the computer’s response).
- **L**oop back to step 1 (to continue the conversation).
- “cntl-c” interrupts program
- “cntl-d” soft reset

Open REPL in MU



Adafruit Circuit Playground Express



CPX: “It’s in there!”

10 x mini NeoPixels, each one can display any color
1 x Motion sensor (LIS3DH triple-axis accelerometer with tap detection, free-fall detection)
1 x Temperature sensor (thermistor)
1 x Light sensor (phototransistor). Can also act as a color sensor and pulse sensor.
1 x Sound sensor (MEMS microphone)
1 x Mini speaker with class D amplifier (7.5mm magnetic speaker/buzzer)
2 x Push buttons, labeled A and B
1 x Slide switch
1 x Infrared receiver and transmitter
8 x alligator-clip friendly GPIO pins: I2C, UART, ADC, PWM, capacitive touch, DAC
Green "ON" LED so you know its powered
Red "#13" LED for basic blinking
Reset button
ATSAMD21 ARM Cortex M0 Processor, running at 3.3V and 48MHz
2 MB of SPI Flash storage, used primarily with CircuitPython to store code and libraries.
MicroUSB port for programming and debugging
USB port can act like serial port, keyboard, mouse, joystick or MIDI!

Keeping it Simple

- Pin 13 LED (Red)
- Neopixels (RGB LEDs)
- Microphone

Introduction to Python

- Compatible with major platforms and systems
- Dynamic typing and binding
- Several sequence types
  - Strings, Lists, Tuples, Dictionaries, Sets
- Powerful subscripting (slicing)
- Functions are independent entities
- Exceptions
- Simple object system
- Iterators and generators
- Asyncio (coroutines)
- Threading

Interpreted vs Compiled

- Interpreted (Python, Javascript, Perl, PHP)
  - Text processed at run-time
  - Interactive environment
  - Many errors not detected until code is run
  - Requires more resources than compiled: slower, more memory
- Compiled (C/C++, Arduino, Rust, Java(bytecode), C#(bytecode))
  - Edit - > Compile - > Link - > Run
  - Many errors caught at compile time
  - Efficient: fast, no interpreter in memory, closer to hardware

Static vs Dynamic Typing

- C/C++, Java: statically typed
  - Variables are declared to refer to objects of a given type
  - Methods use type signatures to enforce contracts
- Javascript, Python : Dynamically typed
  - Variables come into existence when first assigned to
  - A variable can refer to an object of any type
  - All types are (almost) treated the same way
  - Main drawback: type errors are only caught at run-time

Python Basics

- Python language syntax
- Data types and operators
- Variables
- Tuples,lists,dictionaries
- Control flow
- Functions
- Modules and Libraries

Just enough Python to understand Code

- Indentation matters to the meaning of the code:
  - Block structure indicated by indentation
- “import” allows access to libraries and modules
- The first assignment to a variable creates it.
- Variable types don’t need to be declared.
- Python figures out the variable types on its own.
- Assignment uses = and comparison uses ==.
- For numbers + - * / % are as expected.
- Special use of + for string concatenation.
- Use of “{}”.format(var) for string formatting
- Logical operators are words (and, or, not) not symbols
- Simple printing can be done with print().

CircuitPython help

- help(“item”) will normally list extensive information, not in CircuitPython
- help(“modules”) will list modules
- Some editors/IDEs provide context sensitive help
- I recommend keeping a browser open to relevent documentation:
  - https://circuitpython.readthedocs.io/en/latest/docs/index.html

Use <module>.<tab> to get help

```
>>> from adafruit_circuitplayground.express import cpx
>>> cpx.
__qualname__    temperature    touch_A6       detect_taps
_sine_sample    button_a       button_b       _generate_sample
__init__        tapped         switch         play_tone
stop_tone       adjust_touch_threshold        pixels
_touch          red_led        start_tone     play_file
light           shake          touch_A7       touch_A4
touch_A5        __module__     touch_A2       touch_A3
touch_A1        acceleration
```

Import

- Import allows access to libraries and modules:
  - “import time” Everything in “time” time.<func>
  - “From <module> import *” <func> Careful !!!
  - “from adafruit_circuitplayground.express import cpx” cpx.<func>
  - “from time import localtime” only localtime() imported

Try different variations in REPL !!!

```
>>> import time
>>> time.<tab>
__name__        monotonic       sleep           struct_time
localtime       mktime          time
```

Whitespace is meaningful in Python

- indentation and placement of newlines.
- Use a newline to end a line of code.
- Use \ when must go to next line prematurely.
- No braces { } to mark blocks of code in Python…, no ; to end statements
- Use consistent indentation instead.
- The first line with less indentation is outside of the block.
- The first line with more indentation starts a nested block
- Often a colon appears at the start of a new block.

Comments

- Start comments with # – the rest of line is ignored.
- Can include a “””documentation string””” as the first line of any new function or class that you define. Triple double quotes.
- The development environment, debugger, and other tools use it: it’s good style to include one.
- def my_function(x, y):
  “””This is the docstring. This function does blah blah blah.”””
  # The code would go here...

Basic Datatypes

- Boolean
  - Can only be True or False
- Integers (default for numbers)
  - z = 5 // 2 # Answer is 2, integer division.
  - z = 5 / 2 # Answer is 2.5, floating point
- Floats
  - x = 3.456
- Strings
  - Can use “” or ‘’ to specify: “abc” ‘abc’ (Same thing.)
  - Unmatched can occur within the string: “matt’s”
  - Use triple double-quotes for multi-line strings or strings than contain both ‘ and “ inside of them: “””a‘b“c”””

True and False

True and False are constants

In CircuitPython True is normally “on” and False is normally “off”

Other values are treated as equivalent to either True or False when used in conditionals:

- False: zero, None, empty containers
- True: non-zero numbers, non-empty objects
- See PEP 8 Style Guide for the most Pythonic ways to compare
  - https://www.python.org/dev/peps/pep-0008/

Comparison operators: ==, !=, <, <=, etc.

- X == Y
  - X and Y have same value
- X is Y :
  - X and Y refer to the exact same object

Assignment

- Binding a object in Python means setting a name to hold a reference to some object.
- Assignment creates references, not copies (like most languages)
- A name is created the first time it appears on the left side of an assignment expression: x = 3
- An object is deleted (by the garbage collector) once it becomes unreachable.
- Names in Python do not have an intrinsic type.
  - Objects have types.
- Python determines the type of the reference automatically based on what data is assigned to it.

if Statements

- if x == 3:

  print "X equals 3."

  elif x == 2:

  print "X equals 2."

  else:

  print "X equals something else."

  print "This is outside the ‘if’."

- Note:

  Use of indentation for blocks

  Colon (:) after Boolean expression

- Python doesn’t have a “switch” statement. Why? Just because… no good reason

Variables, Comments, Output

```
X = 100    # pound starts a comment, x is an integer

y = ‘Hello’ # y is a string
z = 3.14    # z is a floating point number

if y == ‘Hello’:
     x = x + 1
     y = y + “  World” # String concatenation

print(x)
print(y)
print(“pi = {}”.format(z))

type(z)  # type(<name>) will tell you the type of object a name is bound to
```

Try it in REPL !

Structure of a CircuitPython Program

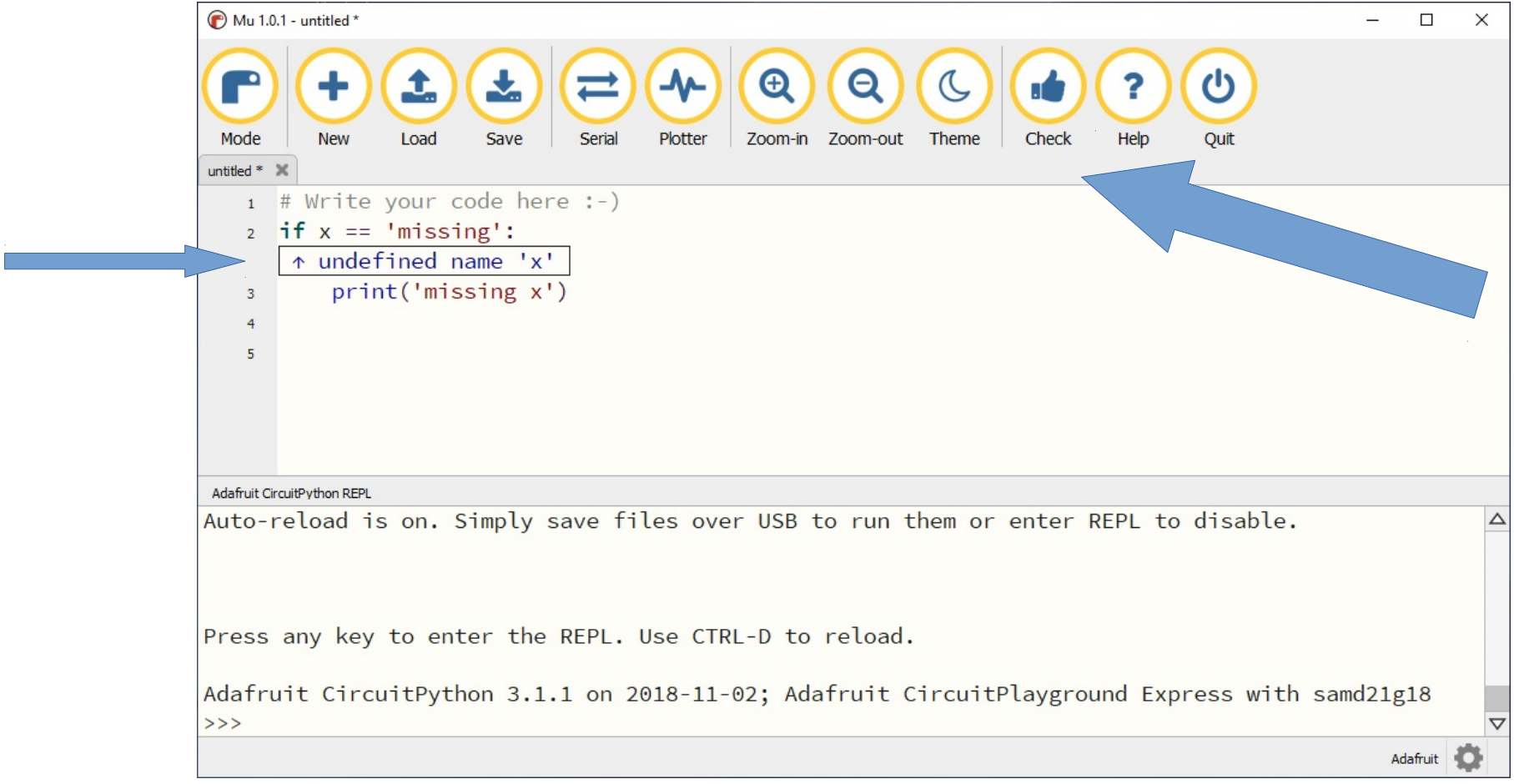
```
import time
import <some stuff>

<define some functions>

While True:          # Note: Infinite loop !
      <do stuff here>
```

Typically “time” is imported so the time.sleep(sec) can “pause” the program

Create main.py on CPX

- In Mu create a new file:
  Click the New button in the top left
- Save as “main.py” to CIRCUITPYI

Use Check to find errors



Create your first program

- Tips:
  - spacing and tabs define structure in Python
  - After an if, function def, or loop, the following line of code MUST, be indented by a tab
  - DO NOT use spaces instead of tabs or you will be in a world of head ache
  - Colon “:” signals the start of a new block
- Type the following code into your editor:

```
import time

from adafruit_circuitplayground.express import cpx

while True:
    cpx.red_led = True
     time.sleep(0.5)
    cpx.red_led = False
    time.sleep(0.5)
```

Running a file on CPX

- The code will run anytime you save a file as “main.py” in the root directory of your “CIRCUITPY” drive
- This file will run whenever you power on or reset your CPX
- To stop the program open REPL, enter <control-c>
- To restart, enter <control-d>

Sequence Types

- 1. Tuple
- A simple immutable ordered sequence of items
- Immutable: a tuple cannot be modified once created....
- tu = (23, ‘this is’, 4.56, (2,3), ‘a tuple’)
- 2. Strings
- Immutable
- Conceptually very much like a tuple of characters
- 3. List
- Mutable ordered sequence of items of mixed types
- li = [“This is”, 34, 4.34, 23, “a list”]

Accessing elements

We can access individual members of a tuple, list, or string

- using square bracket “array” notation.
- Note that all are 0 based…
- >>> tu = (23, ‘abc’, 4.56, (2,3), ‘def’)
- >>> tu[1] # Second item in the tuple.
- ‘abc’
- >>> li = [“abc”, 34, 4.34, 23]
- >>> li[1] # Second item in the list.
- 34
- >>> st = “Hello World”
- >>> st[1] # Second character in string.
- ‘e’

Understanding Python Reference Semantics

- Key concept: mutable (changeable) vs immutable (fixed) objects
- Assignment manipulates references (binds)
  - x = y does not make a copy of the object y references
  - x = y makes x reference the object y references
- Very useful; but beware!
  - Example:

    ```
    >>> a = [1, 2, 3] # a now references the list [1, 2, 3]
    >>> b = a # b now references what a references
    >>> a.append(4) # this changes the list a references
    >>> print b # if we print what b references,
    [1, 2, 3, 4] # SURPRISE! It has changed…
    ```

  - To Copy: b = a[:] or b = a.copy()

For Loops

For Loops 1

For-each is Python's only form of for loop

A for loop steps through each of the items in a collection type, or any other type of object which is "iterable"

for <item> in <collection>:
<statements>

If <collection> is a list or a tuple, then the loop steps through each element of the sequence.

If <collection> is a string, then the loop steps through each character of the string.

```
for someChar in "Hello World":
print someChar
```

For loops continued...

for <item> in <collection>:
<statements>
    <item> can be more complex than a single variable name.
    If the elements of <collection> are themselves collections, then <item> can match the structure of the elements. (We saw something similar with list comprehensions and with ordinary assignments.)

```
for (x, y) in [(a,1), (b,2), (c,3), (d,4)]:
print x
```

Try out some lists in REPL

```
goo = ["candy", "mud"]
print("Gooey stuff={}".format(goo))
for x in goo:
    print("{} is gooey".format(x))
```

Playing with Neopixels

- The CPX has 10 Neopixel LEDs (WS2812)
- Colors are specified as (Red, Green, Blue)
  - Brightness levels: 0 to 255
  - Red = (255,0,0), Purple = (100,0,100), Yellow=(255,255,0)
- More here:
  - https://learn.adafruit.com/adafruit-neopixel-uberguide/the-magic-of-neopixels

Try working with Neopixels in REPL

```
>>> import board
>>> import neopixel
>>> pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=.2)
>>> pixels[0] = (0,10,0)
>>> pixels[1] = (10,0,0)
```

Defining Functions

Function definition begins with def Function name and its arguments.
‘return’ indicates the
value to be sent back to the caller.

```
def get_final_answer(filename):
    """Documentation String"""
    line1
    line2
    return total_counter

outside = ‘of function’
```

First line with less
indentation is considered to be
outside of the function definition.

Function Gotchas

- All functions in Python have a return value, even if no return line inside the code.
- Functions without a return return the special value: None.
- There is no function overloading in Python.
- Two different functions can’t have the same name, even if they have different arguments.
- Functions can be used as any other data type.
  - They can be:
    - Arguments to function
    - Return values of functions
    - Assigned to variables
    - Parts of tuples, lists, etc

For loops and the range() function

For loops and the range() function

We often want to write a loop where the variables ranges over some sequence of numbers. The range() function returns a list of numbers from 0 up to but not including the number we pass to it.

range(5) returns 0,1,2,3,4

So we can say:

```
for x in range(5):
print x
```

(There are several other forms of range() that provide variants of this functionality…)

xrange() returns an iterator that provides the same functionality more efficiently

while Loops

```
>>> x = 3
>>> while x < 5:
print x, "still in the loop"
x = x + 1
3 still in the loop
4 still in the loop
>>> x = 6
>>> while x < 5:
print x, "still in the loop"
>>>
```

Load neopixel.py

- Close any files open in Mu
- Make a copy of neopixel.py
- Rename it main.py
- Copy it to CPX

Experiment with Neopixels

- Try:
  - Changing the timing
  - Changing the colors
  - Changing the number of Neopixels modified

CPX Microphone and Math

- Set the Neopixels color based on volume
- Read samples from microphone
- Calculate RMS
- Map values to a range of colors
- Set and display color

Python array

- Arrays create a memory buffer
- The size and ‘c’ data type are specified at creation
- Arrays are contiguous memory that can be used to exchange data with entities outside of Python… like audio hardware...

Arrays continued

To create an array:
import array
Call the create method: array.array(typecode[, initializer])

samples = array.array('H', [0] * NUM_SAMPLES)

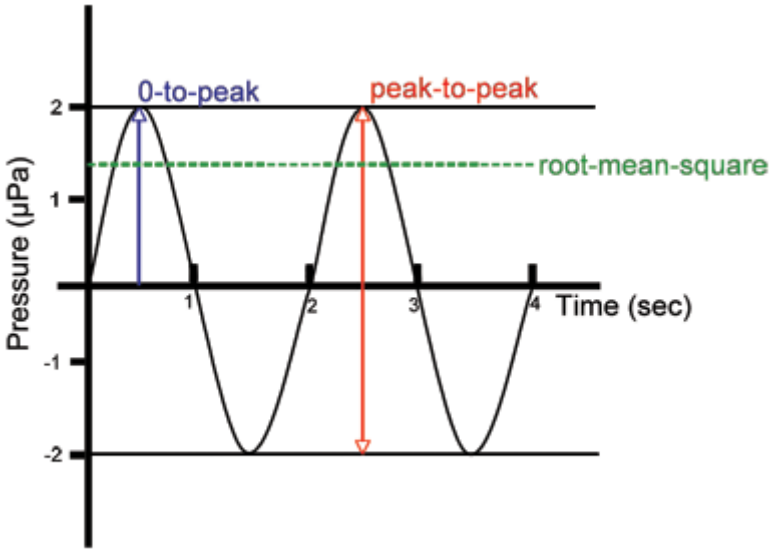
What it does:
‘H’ = allocate unsigned shorts (2 bytes)
Initialize the array with NUM_SAMPLES elements set to zero
“[0] * NUM_SAMPLES” creates a list of zeros with NUM_SAMPLES elements

Audio RMS

We only use two “math” functions for calculate the audio RMS value

math.pow(x, y) and math.sqrt(x)

$$rms\ pressure = \sqrt{(\rho^2)}_{average}$$



Details: https://dosits.org/science/advanced-topics/introduction-to-signal-levels/

How to calculate RMS

Steps for calculating rms pressure	Sample calculation
1. Measure the pressure at points along the sound signal.	0, 2, 0, -2, 0, 2, 0, -2
2. Square the measured pressures.	0, 4, 0, 4, 0, 4, 0, 4
3. Average the squared pressures.	(0+4+0+4+0+4+0+4)/8 = 2
4. Take the square root of the average of the squared pressures.	√2 = 1.4

So how loud is it?

- Sound levels vary over a wide range
- Human perception of sound is exponential (or logarithmic depending on which way you look at it)
- Sound levels are measured in decibels:
  - 0 db = threshold of hearing
  - 10 db = 10 times as powerful, humans perceive volume doubling
  - 20 db = 100 times as powerful
  - 30 db = 1000 time

Formula for decibels:

$$I(db) = 10log_{10}\left(\frac{I_{Sound}}{I_{Reference}}\right)$$

Load MusicLight.py

- Close any files open in Mu
- Make a copy of MusicLight.py
- Rename it main.py
- Copy it to CPX

Putting it all together

```
import array
import math
import time
import audiobusio
import board
import neopixel
```

Putting it all together 2

```
# Exponential scaling factor.
# Should probably be in range -10 .. 10 to be reasonable.
CURVE = 2
SCALE_EXPONENT = math.pow(10, CURVE * -0.1)

NUM_PIXELS = 10

# Number of samples to read at once.
NUM_SAMPLES = 160
```

Putting it all together 3

```
def mean(values):
    return sum(values) / len(values)

# Remove DC bias before computing RMS.
def normalized_rms(values):
    minbuf = int(mean(values))

    # remove DC bias and square samples
    samples = []
    for sample in values:
        samples.append(float(sample - minbuf) * (sample - minbuf))

    samples_sum = sum(samples)
    return math.sqrt(samples_sum / len(values))
```

Putting it all together 4

```
# Scale input_value between output_min and output_max, exponentially.
def log_scale(input_value, input_min, input_max, output_min, output_max):
    normalized_input_value = (input_value - input_min) / \
                             (input_max - input_min)
    return output_min + \
        math.pow(normalized_input_value, SCALE_EXPONENT) \
        * (output_max - output_min)
```

Putting it all together 5

```
# Restrict value to be between floor and ceiling.
def constrain(value, floor, ceiling):
    return max(floor, min(value, ceiling))

def bgr_vol_2_color(val):
    # Input a value 0 to 255 to get a color value.
    # The colours are a transition b - g - r.

    # Make sure val is in range
    val = constrain(val, 0, 255)
    if val == 0:
        return (0,0,0)

    if val < 127:
        return (0, int(val * 2), int(255 - val * 2))
    else:
        val -= 127
        return (int(val * 2), int(255 - (val * 2)), 0)
```

Putting it all together 6

```
def do_lvl_3_color_meter(snd_level, pixels):
    # Light up pixels that are below the scaled and interpolated magnitude.
    pixels.fill(bgr_vol_2_color(snd_level))
    pixels.show()
```

Putting it all together 7

```
def main():

    pixels = neopixel.NeoPixel(board.NEOPIXEL, NUM_PIXELS,
                               brightness=0.1, auto_write=False)
    pixels.fill((100, 0, 100))
    pixels.show()
    time.sleep(2)
```

Putting it all together 8

```
# For Circuitpython 3.0 and up, "frequency" is now called "sample_rate".
    # Comment the lines above and uncomment the lines below.
    mic = audiobusio.PDMIn(board.MICROPHONE_CLOCK, board.MICROPHONE_DATA,
                           sample_rate=16000, bit_depth=16)

    # Record an initial sample to calibrate. Assume it's quiet when we start.
    samples = array.array('H', [0] * NUM_SAMPLES)
    mic.record(samples, len(samples))
```

Putting it all together 9

```
# Set lowest level to expect, plus a little.
 input_floor = normalized_rms(samples) + 20
 # OR: used a fixed floor
 # input_floor = 50
 # You might want to print the input_floor to help adjust other values.
 # print(input_floor)
 # Corresponds to sensitivity:
 #     lower means more pixels light up with lower sound
 # Adjust this as you see fit.
 input_ceiling = input_floor + 500
```

Putting it all together 10

```
while True:
    mic.record(samples, len(samples))
    magnitude = normalized_rms(samples)
    # You might want to print this to see the values.
    # print(magnitude)

    # Compute scaled logarithmic reading in the range 0 to number of colors
    c = log_scale(constrain(magnitude, input_floor, input_ceiling),
                  input_floor, input_ceiling, 0, 255)

    # Filter out noise, brute force, can be improved
    if c < 50:
        c = 0

    do_lvl_3_color_meter(c, pixels)
```

Putting it all together 11

```
if __name__ == '__main__':
    main()
```

This statement allows the code to be reused as a module. It only executes main() if it is not “import”ed

Details in the next class…. Or you can search online for “python special variables”

Python Quirks

- Python binds Names to Objects
  - Names bound to objects NOT EQUAL Variables pointers to memory
- Mutable vs Immutable
  - Variable reference changes shared value (Mutable) or doesn't (Immutable)
  - Using mutable as default value: (first call numbers created, following just appended)
    ```
    def foo(numbers=[]):
        numbers.append(9)
        return numbers
    ```
  - a = [2, 4, 8] (mutable), b = a (b points to a), Copy: b = a[:] or b = a.copy()
- Scoping: global can be referenced, but assignment creates local variable
- Creating count-by-one errors on loops, a = list(range(1, 11)), yields [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
- Python is case sensitive
- Python "array" is needed to build buffer, "real" memory array
- Equivalency test "is" vs "=="
  - Use "==", example:
    ```
    x = 256
    print(x is 256)  # prints True
    y = 257
    print(y is 257)  # prints False
    ```

Python Recommended Reading

- On-line Python tutorials
  - The Python Tutorial (http://docs.python.org/tutorial/)
    - Dense but more complete overview of the most important parts of the language
  - PEP 8- Style Guide for Python Code
    - http://www.python.org/dev/peps/pep-0008/
    - The official style guide to Python, contains many helpful programming tips
  - U of Waterloo Beginners Python Tutorial: https://cscircles.cemc.uwaterloo.ca/using-this-website/
- Many other books and on-line materials

Credits

Heavily based on presentations by
Matt Huenerfauth (Penn State)
Guido van Rossum (Google)
Richard P. Muller (Caltech)
Adafruit CircuitPython documentation
Mu Editor documentation
Adafruit Playground Sound Meter
...