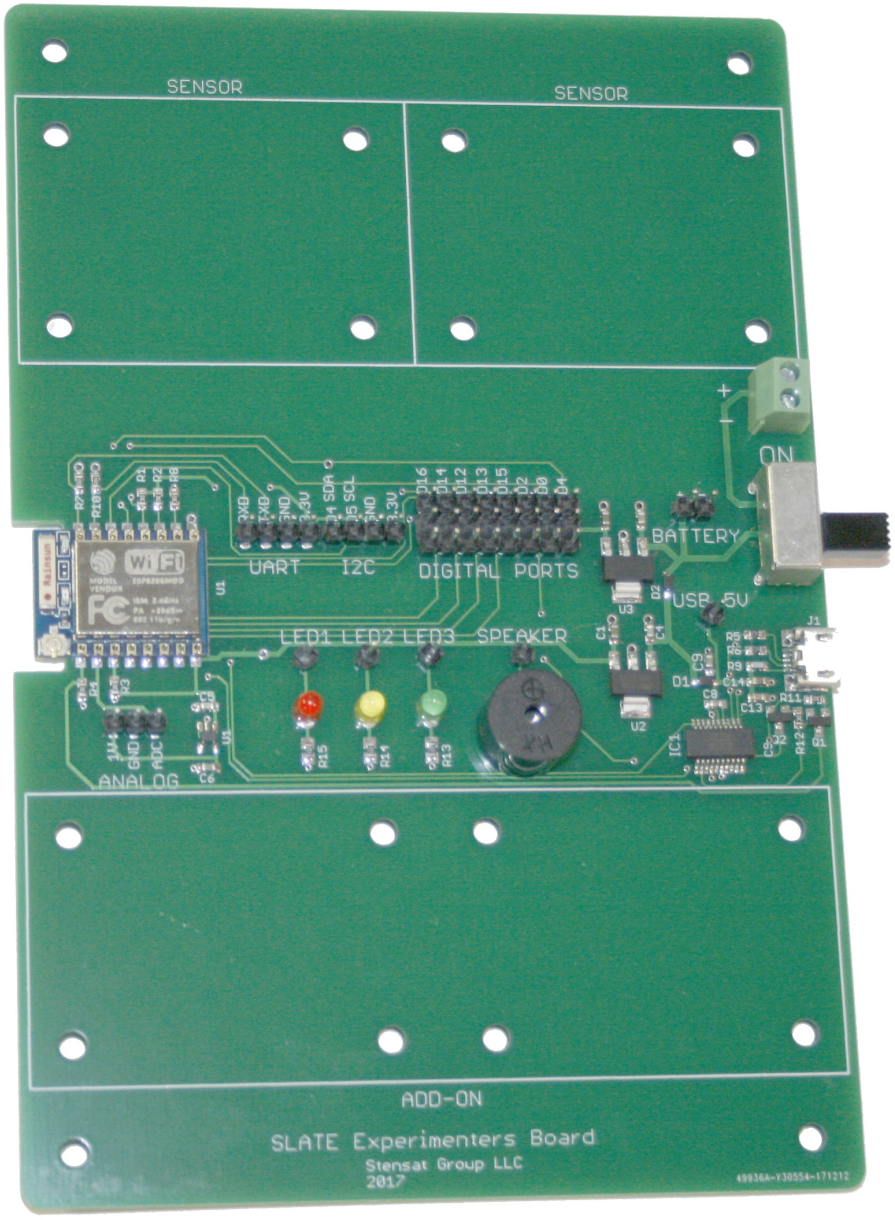
Sten-SLATE ESP Kit

Python



Stensat Group LLC,

Python

Python IDE

Python is an interpreted language. This means that a compiler is not used to generate a executable file. Instead, the python program interprets the written program directly. This lesson assumes basic knowledge of python. You can learn the basics of python at www.learnpython.org

There are a few ways to write python programs. The method shown here is very simple and straight forward. This lesson will introduce you to the pygame library and the socket library that will be used to interact with the SYST101 and SYST395 kits.

Both Windows and Mac OS X will be covered.
For Windows, it is assumed Windows 10 or Windows 7 is being used. Python version 3.8.7 will be used.

Go to www.python.org and select Downloads. Locate the latest Python version 3.8.

Scroll down to Files and select **Windows x86-64 executable installer**.

Once downloaded, start the installer program. At the start windows, select **Add Python 3.8 to Path** then click on **Install Now.** Python will now install and show up in the Start menu. Close the window when the installation has completed.

In the start up menu, select **Windows Powershell** and select **Windows Powershell**. Do not select the ISE version.

In the Powershell, type **pip3 install tk** and press **Enter**. This will install the TK library for Python. Once completed, exit **Windows Powershell**. This completes the installation. To start Python, select **Python IDLE** in the **Start** menu.

Python For Mac OS X

It is assumed the latest Mac OS X release is being used. Go to
www.python.org/downloads/mac-osx

Locate **python 3.8.2** and select **Mac OS 64-bit installer**.

The installer will be downloaded. Once downloaded, double click on the program and it will go through the installation process. Once installed, the Python 3.8 will be located in the Application folder. Open the Python 3.8 folder and you can double click on IDLE.app to start Python.

To install the TK library, open the terminal application. In the terminal application enter the following:

pip3 install tk and press **Enter.**

The library will be installed.

Python IDLE

The Python IDLE is a python shell. You can enter python commands and and they will execute immediately.

Start Python IDLE. At the **>>>** prompt, enter the command:

```
>>>print(“Hello world”)
Hello world
```

Equations can be entered and Python will generate the answer:

```
>>>5+3
8
```

Python uses indentation to define a block of code. This means all code indented after an if statement, while statement or other conditional statement will be executed.

```
if 10 > 5:
     print(“10 is larger”)
     print(“Second line”)
```

After entering the above, press enter again. The commands will execute.

Here is a while loop:

```
while 1:
     print(“this is a loop”)
```

You can stop the while loop by pressing <Ctrl> and C keys.
The IDLE will also let you create new programs using an editor. Select menu **File** and **New**. An editor will open and you can enter a python program. Try the program above with the while loop. Save the file. You will be prompted for a name. By convention, use .py for the file name extension. Once saved, select menu **Run** and **Run Module**. Press CTRL and C to stop the program. Now you should have a basic understanding on using the IDLE.

Basic Python

Variables are used to store values or strings. You do not need to declare what type of variable it is. It is automatically determined when a value is assigned. Variables can change types by assignment. Variables must start with a letter. It cannot start with a number. Variables can only be made up of letters, numbers and _.

```
a = 5
b = “Fred”
print(a)
print(b)
print(a,b)
```

```
a = 5
print(a)
a = “Fred”
print(a)
```

Math operations are simple in Python. Just use equations.

Notice the # sign and the text to the right. This is a comment. Comments are useful to explain what the code is supposed to do.

```
a = 5 + 6                # addition
b = 12.43                # floating point
c = b – 3.14             # subtraction
dog = 45.834             # assignment
cat = dog * b            # multiplication
dd = cat / a       # division
```

Variable and values can be compared with each other. Python provides multiple ways to do comparisons

== is equal

!= is not equal

> greater than

< less than

>= greater than or equal

<= less than or equal

Enter the code into a program and run it. You can change the values of the variables and see how execution changes.

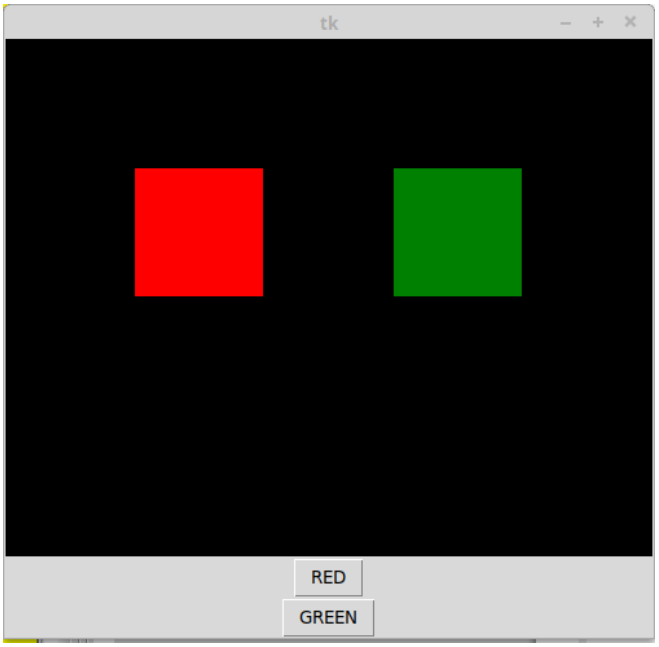
```
a = 45
b = 32
if a > b:
     print(“a is greater than b”)
if a < b:
     print(“a is less than b”)
if a == b:
     print(“a is equal to b”)
if a != b:
     print(“a is not equal to b”)
```

Rules of Python Coding

When indenting, it is very critical to use the same number of spaces otherwise the code will not be interpreted correctly. Indented code is used to identify code associated with **while**, **for**, **if** and functions. Tabs and spaces are not interchangeable and will cause program errors or incorrect code execution. Using the IDLE editor will help keep the indentation uniform. Problems can occur if you import code from elsewhere and tabs and spaces are not used consistently.

First GUI Program

The Tk or Tkinter library will be used for creating a graphical user interface in the following example programs. Tkinter treats all user interfaces called widgets and graphical components as objects. Widgets includes buttons, pull down menus. radio buttons, text entry, selectors, sliders and canvases. Each widget can be customized in size, color, etc.



The first program will introduce buttons and the canvas. The buttons are clicked on with a mouse and the canvas is an area to render graphics.

The black region is the canvas and the two buttons are stacked below the canvas.

When clicking on either button, the click action will execute the corresponding function.

The two rectangles rendered in the canvas are also objects. They can be created and assigned to a unique variable. This will allow the rectangles to be modified later. In this example, the color is modified.

Note that the graphic objects are rendered in order of code execution. A graphic object can be on top of another. Modifying the object does not change which is on top.

Widgets are stacked in order of execution from top to bottom using the **pack()** function. There are other window managers to allow more flexible layouts.

The Tkinter library is a event based library in that functions are called when events occur such as clicking on a button or entering text in a text widget or doing changing any widget. You will notice that all the python programs using Tkinter end with mainloop(). This function is where Tkinter monitors the widgets for event triggers such as clicking on a button. No other code can execute after mainloop().

In this lesson some of the widgets will be used. Others can be found online.

Check out this site for more widgets.

https://www.dummies.com/programming/python/using-tkinter-widgets-in-python/

TK Library

Tk is a library that allows you to create a graphical user interface. It includes many different widgets such as buttons, menus, text entry, lists and so on. It also provides a canvas widget that allows for rendering graphics.

The code to the right creates a program with two buttons and a canvas to render graphics. When any of the buttons are pressed, a corresponding function is called.

Two rectangles will be created and rendered in the canvas area. When you click on a button, the color of the corresponding rectangle will change color.

At the top, the Tk library also called tkinter is imported. The way it is imported allows the functions to be called more easily.

top = Tk() creates the window for the GUI. The variable **top** relates to the window created. Multiple windows can be created and each will need their own name.

Next, two variables are declared to indicate the color state of the rectangles.

```
from tkinter import *

top = Tk()

toggle_red = 0
toggle_green = 0

c = Canvas(top,bg='black',height=400,width=800)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
    else:
        color = 'gray'
        toggle_red = 0
    c.itemconfig(rrect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
    else:
        color = 'gray'
        toggle_green = 0
    c.itemconfig(grect,fill=color)

rrect = c.create_rectangle(100,100,200,200,fill ='gray')
grect = c.create_rectangle(300,100,400,200,fill = 'gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack()
gb.pack()
top.mainloop()
```

TK Library

Next, the canvas widget is created with a black background, with 800 pixels in the X direction and 400 pixels in the Y direction. The first argument specifies the window the canvas is to be placed.

Skip the two functions for now.

Two rectangles are created and given variable names **rrect** and **grect**. Graphics rendered are objects and can be later modified. The numbers in the arguments are the top left corner X and Y and the bottom right corner X and Y locations. The **fill** parameter sets the color of the rectangle. Notice the **c.** in front of the create function. This makes the rectangles get rendered in the canvas assigned to variable **c**. You can have more than one canvas.

Two buttons are created. The first argument is the window to place the buttons. The argument **text** sets the text in the button. The argument **command** tells what function to execute when the button is clicked.

```
from tkinter import *

top = Tk()

toggle_red = 0
toggle_green = 0

c = Canvas(top,bg='black',height=400,width=800)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
    else:
        color = 'gray'
        toggle_red = 0
    c.itemconfig(rrect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
    else:
        color = 'gray'
        toggle_green = 0
    c.itemconfig(grect,fill=color)

rrect = c.create_rectangle(100,100,200,200,fill ='gray')
grect = c.create_rectangle(300,100,400,200,fill = 'gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack()
gb.pack()
top.mainloop()
```

TK Library

At the bottom of the code, there are three lines that tell the window manager to display the widgets. The **pack()** function will pack the widgets in the window in a vertical direction from top to bottom in order of the code.

The last line starts the loop that operates the widgets. The button click will only be detected when **top.mainloop()** is executed. The program stops when you close the window.

Nothing executes after **mainloop()**. This function manages the widgets and calls the related functions when the widgets are activated. This is a type of event based programming where actions occur when events are detected.

```
from tkinter import *

top = Tk()

toggle_red = 0
toggle_green = 0

c = Canvas(top,bg='black',height=400,width=800)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
    else:
        color = 'gray'
        toggle_red = 0
    c.itemconfig(rrect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
    else:
        color = 'gray'
        toggle_green = 0
    c.itemconfig(grect,fill=color)

rrect = c.create_rectangle(100,100,200,200,fill ='gray')
grect = c.create_rectangle(300,100,400,200,fill = 'gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack()
gb.pack()
top.mainloop()
```

TK Library

Functions in python are declared with the **def** statement. The name of the function and ends in colon. All instructions for the function must be indented as shown.

The statement global tells python the variable used is not localized but the same one used in the main program. Python will automatically declare any variables created or referenced inside a function as local. This means you can use the same variable name in the main part of the code and in the function and they will be completely independent. The variable related to Tk are global by default.

The function checks the state of the toggle_red or green to determine the color of the rectangle. If it is zero, the color is set to red or green and the toggle variable is changed. This allows tracking of the state of the rectangle color. The functions are called only when the button is clicked.

The function **c.itemconfig()** lets you change the state of any graphic object you put into the canvas. In this example, the color of the rectangle is changed.

Save the program and call it buttons.py. It will be used later.

```
from tkinter import *

top = Tk()

toggle_red = 0
toggle_green = 0

c = Canvas(top,bg='black',height=400,width=800)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
    else:
        color = 'gray'
        toggle_red = 0
    c.itemconfig(rrect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
    else:
        color = 'gray'
        toggle_green = 0
    c.itemconfig(grect,fill=color)

rrect = c.create_rectangle(100,100,200,200,fill ='gray')
grect = c.create_rectangle(300,100,400,200,fill = 'gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack()
gb.pack()
top.mainloop()
```

Tk Canvas

In this example, you will use a mouse to draw on the canvas.

There are two functions in this program, one to paint and the other to clear the screen.

top.title() fucntion Changes the title of the window at the top.

The Canvas is created as before. **w.bind()** is used to capture mouse events when the mouse is over the canvas. The first one is **<B1-Motion>** which is used to call the paint function. When the left button is pressed and the mouse is moved, the paint function will be called repeatedly until motion stops and the button is released. The **w.bind("<Button-3>", clear)** is used to detect the click of the right button. The clear function is called any time the right button is clicked while the mouse is over the canvas.

A new widget in the code is the Label. This is used to place a text label in the window. It is placed below the canvas due to the order of the **pack()** functions.

The **paint()** function renders a small red circle each time it is called at the mouse location. The event passes mouse information into the variable event. The event variable is an object that contains the X and Y position of the mouse. Each circle created is an object. At this time, it is not assigned to an object name as in the program on page 12.

w.delete("all") in the clear function deletes all the objects created in the canvas and clears the canvas.

```
from tkinter import *


canvas_width = 500
canvas_height = 300


def paint( event ):
   color = "#ff0000"
   x1, y1 = ( event.x - 1 ), ( event.y - 1 )
   x2, y2 = ( event.x + 1 ), ( event.y + 1 )
   w.create_oval( x1, y1, x2, y2, fill = color )


def clear(event) :
    w.delete("all")

top = Tk()
top.title( "Painting using Ovals" )
w = Canvas(top,
           width=canvas_width,
           height=canvas_height,bg='black')
w.pack()
w.bind( "<B1-Motion>", paint )
w.bind("<Button-3>",clear)


message = Label( top, text = "Press and Drag the mouse to draw" )
message.pack()

mainloop()
```

The link below provides more information on how to link events. Any object such as buttons and labels can have events attached. Even the top object can have events. Events are not limited to the mouse, keyboard inputs can also be detected.

https://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

Tk Canvas

In this example, two new modules are introduced, threading and time.

Threading is the ability to have one or more functions execute simultaneously. Since Tkinter is an event based module, there are times when code is needed to execute after **mainloop()** is executed. Threading will allow a function to execute in parallel with Tkinter **mainloop()**.

The time module provides a function to stop the program for a specified time similar to the **delay()** function in the Arduino software.

```
from tkinter import *
import threading
import time

top = Tk()
c = Canvas(top,width=800,height=600,bg='black')

r = c.create_rectangle(300,10,500,20,fill='yellow')

def stretch():
    while 1:
        for s in range(20,500):
            c.coords(r,300,10,500,s)
            time.sleep(.01)

x = threading.Thread(target=stretch)
x.start()
c.pack()
mainloop()
```

In the code to this page, a canvas is created and a rectangle is created and positioned at the top and center of the canvas. After the function stretch(), a thread is created named x. The argument is the name of the function to execute in parallel. After creating the thread, it is started. The start of the thread must occur after the rectangle is created otherwise the function will try accessing the rectangle object when it doesn't exist and an error will occur stopping the program.

After the thread is started, the canvas is placed in the window using the **pack()** function and the **mainloop()** is called. When run, a window will open and the rectangle will be drawn and it will grow in length downward.

Looking at the stretch() function, the function starts with **while 1:**. This creates an infinite loop since we want the thread to keep running and not just once. The next line is a **for** loop. Variable **s** is assigned a value starting at 20 and loops incrementing **s** by 1 until 500. The loop stops at 499 and exits at 500 so the rectangle never gets stretched to 500. range() is a function that will return a value starting with the first argument and increment by one until reaching the last increment. If a third argument is added, that will be the increment size. An example is:

for s in range(20,500,2):

This will increment s by two from 20 to 500.

In the for loop, **c.coords(r,300,10,500,s)** changes the coordinates of the **r** rectangle. The **c** in **c.coords()** specifies canvas **c**. You can have more than one canvas in a program. They just need different names.

time.sleep() stops the program for the specified number of seconds. Fractional seconds can be specified. The example stops for 10 milliseconds.

Try the program and change the range() and add different steps. Adjust the sleep length.