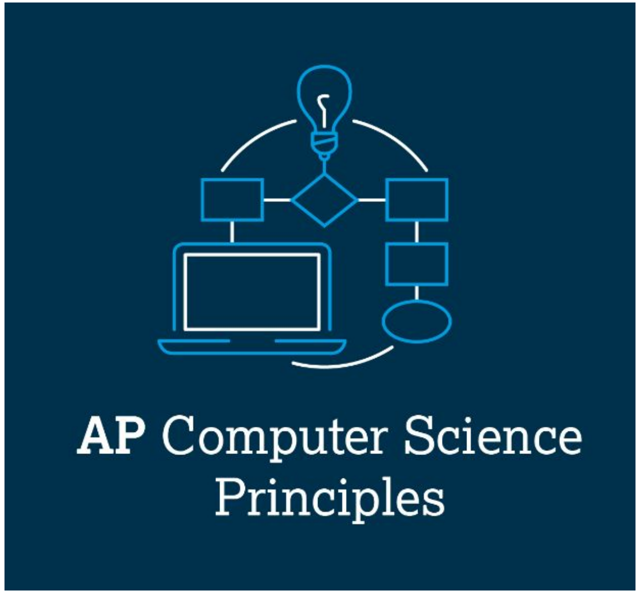
Practice Extra

Running a game multiple times

AP CSP Create Performance Task

Part of the AP Exam is to create a program that meets specific requirements:

- Creates a list
- Uses a list in a meaningful way
- Has a function with a parameter
    - Parameter is used in an if statement
- Function has:
    - If statement
    - Loop

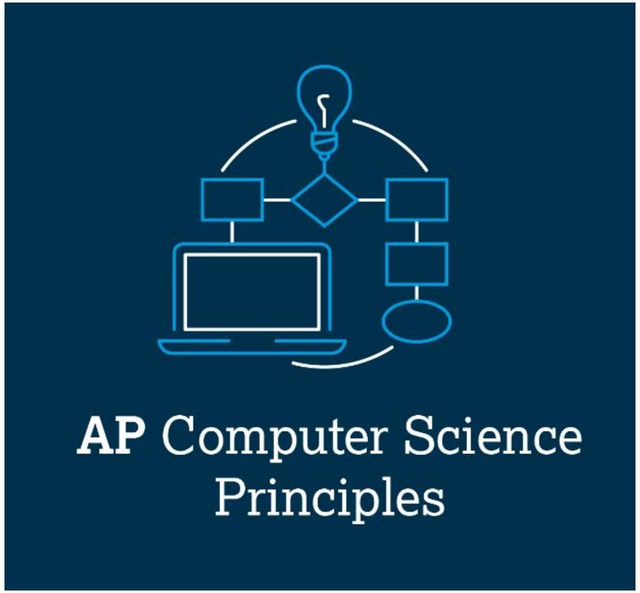


AP CSP Create Performance Task

This Practice Extra is something extra you can add to a program that already meets the requirements, but will make your game program more user-friendly.

For this project, you will:

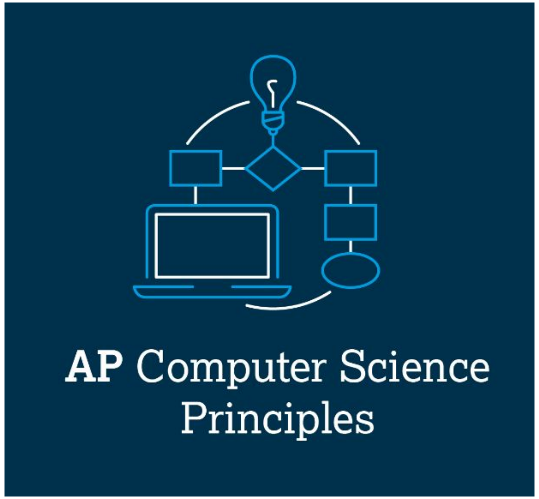
- Learn how to play a game again without restarting the code



AP CSP Create Performance Task

A few of your missions and remixes involve playing a game. Some of the Create PT Practices also involve playing a game.

- For most of these programs, you must restart the code to play again
- Today’s lesson: use a while loop to play again without restarting



Mission 10 - Reaction Time

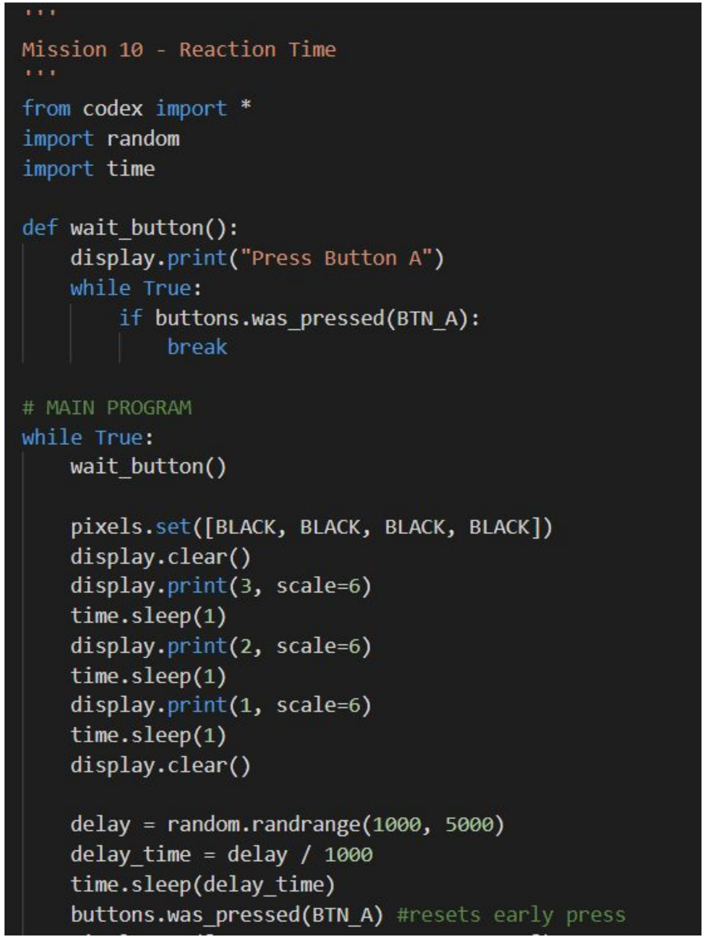
This program calculates the time it takes for the user to press a button in reaction to pixels lighting.

- It is already in a loop
- But … the loop never ends
- In order to stop playing, the user has to manually stop the code
- This is the same kind of problem as having to manually start the code again.
- Learn how to fix the problem in this mission, then apply the solution to other program games

Step #1

Open your project from Mission 10 - “Reaction_time”

- Run the code and make sure it works properly
- The game plays continuously until you stop the program.

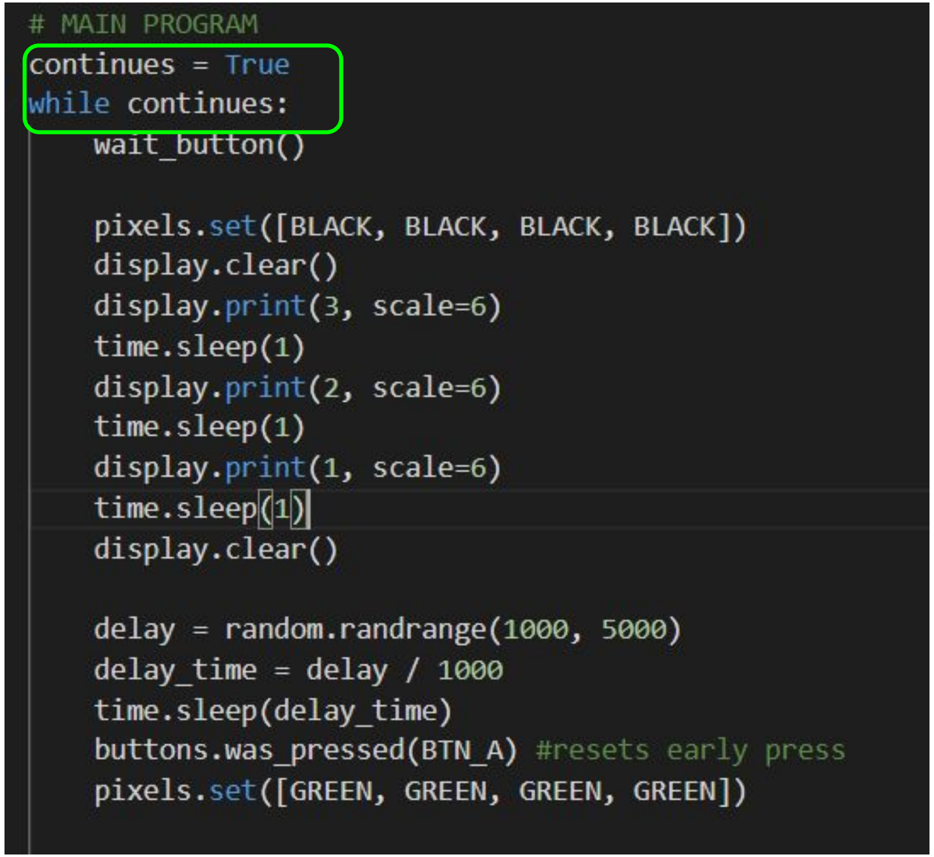


```
'''
Mission 10 - Reaction Time
'''
from codex import *
import random
import time

def wait_button():
    display.print("Press Button A")
    while True:
        if buttons.was_pressed(BTN_A):
            break

# MAIN PROGRAM
while True:
    wait_button()

    pixels.set([BLACK, BLACK, BLACK, BLACK])
    display.clear()
    display.print(3, scale=6)
    time.sleep(1)
    display.print(2, scale=6)
    time.sleep(1)
    display.print(1, scale=6)
    time.sleep(1)
    display.clear()

    delay = random.randrange(1000, 5000)
    delay_time = delay / 1000
    time.sleep(delay_time)
    buttons.was_pressed(BTN_A) #resets early press
```

Step #1

Modify the while loop

- Use a global Boolean variable as the condition to the loop
- Remember: a Boolean variable is either True or False
- Therefore, you don't need a comparison, just the variable

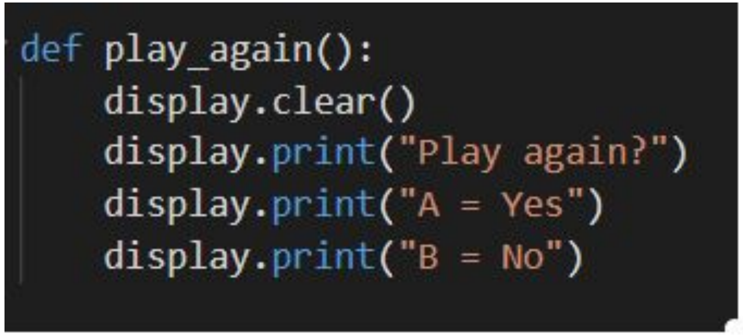


```
# MAIN PROGRAM
continues = True
while continues:
    wait_button()

    pixels.set([BLACK, BLACK, BLACK, BLACK])
    display.clear()
    display.print(3, scale=6)
    time.sleep(1)
    display.print(2, scale=6)
    time.sleep(1)
    display.print(1, scale=6)
    time.sleep(1)
    display.clear()

    delay = random.randrange(1000, 5000)
    delay_time = delay / 1000
    time.sleep(delay_time)
    buttons.was_pressed(BTN_A) #resets early press
    pixels.set([GREEN, GREEN, GREEN, GREEN])
```

Step #1

Add another function

- Create a function that asks the user if they want to play again
- You can use print statements or any other way to display information

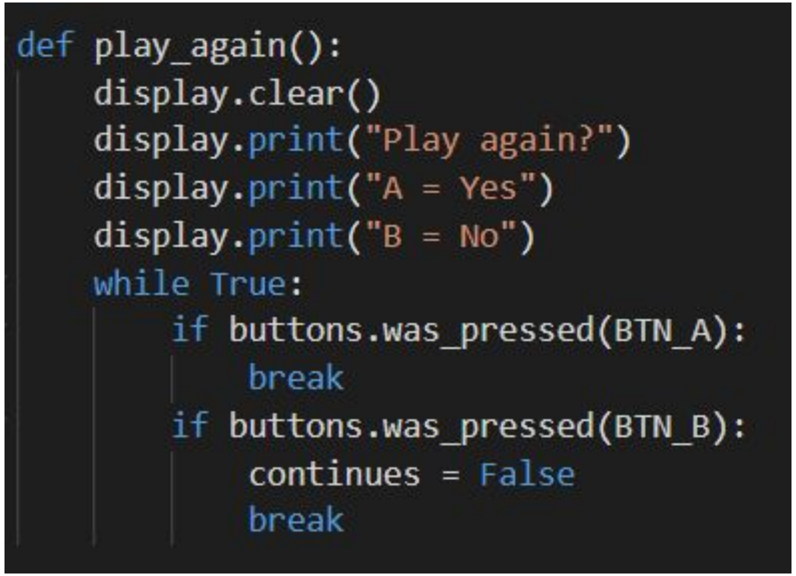


```
def play_again():
    display.clear()
    display.print("Play again?")
    display.print("A = Yes")
    display.print("B = No")
```

Step #1

Add another function

- Now add a while True loop and wait for a button A or button B press.
- What should happen for each one?
- For button A, just break
- For button B, assign **False** to the Boolean variable and then break

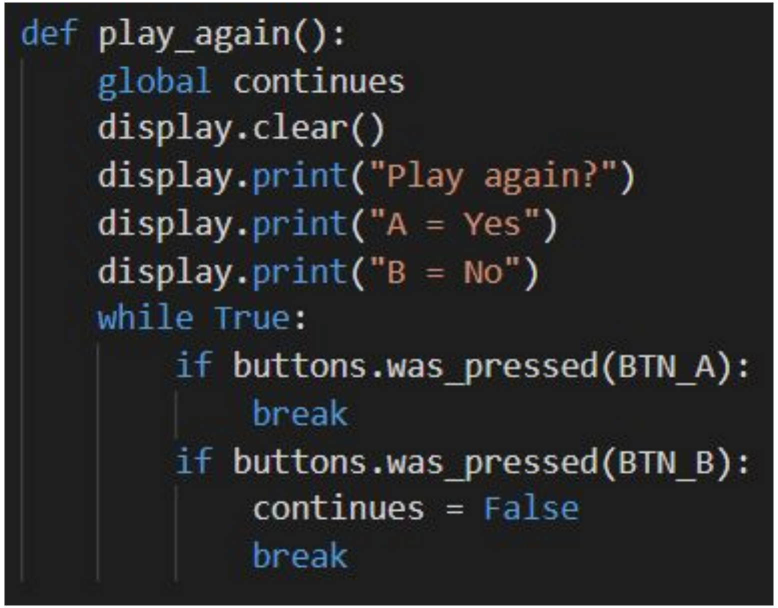


```
def play_again():
    display.clear()
    display.print("Play again?")
    display.print("A = Yes")
    display.print("B = No")
    while True:
        if buttons.was_pressed(BTN_A):
            break
        if buttons.was_pressed(BTN_B):
            continues = False
            break
```

Step #1

Add another function

- Did you notice something about the code for button B?
- **continues** is a global variable, and its value changes
- So, add the code you need to for a global declaration

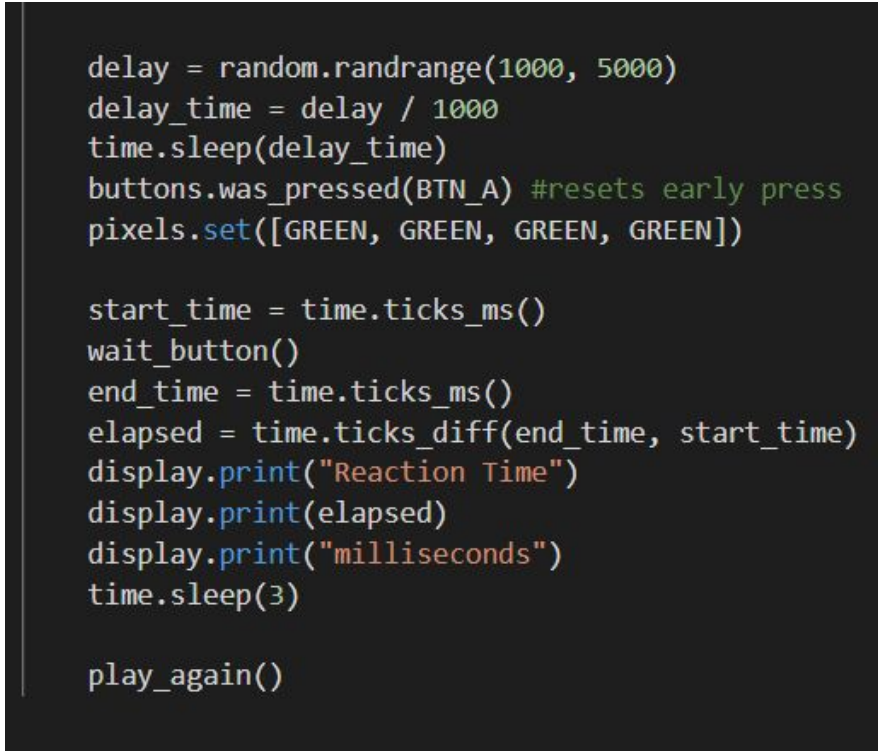


```python
def play_again():
    global continues
    display.clear()
    display.print("Play again?")
    display.print("A = Yes")
    display.print("B = No")
    while True:
        if buttons.was_pressed(BTN_A):
            break
        if buttons.was_pressed(BTN_B):
            continues = False
            break
```

Step #1

Add another function

- Call the function in the main program
- Can you now stop playing the game without manually stopping the code execution?
- Test and debug

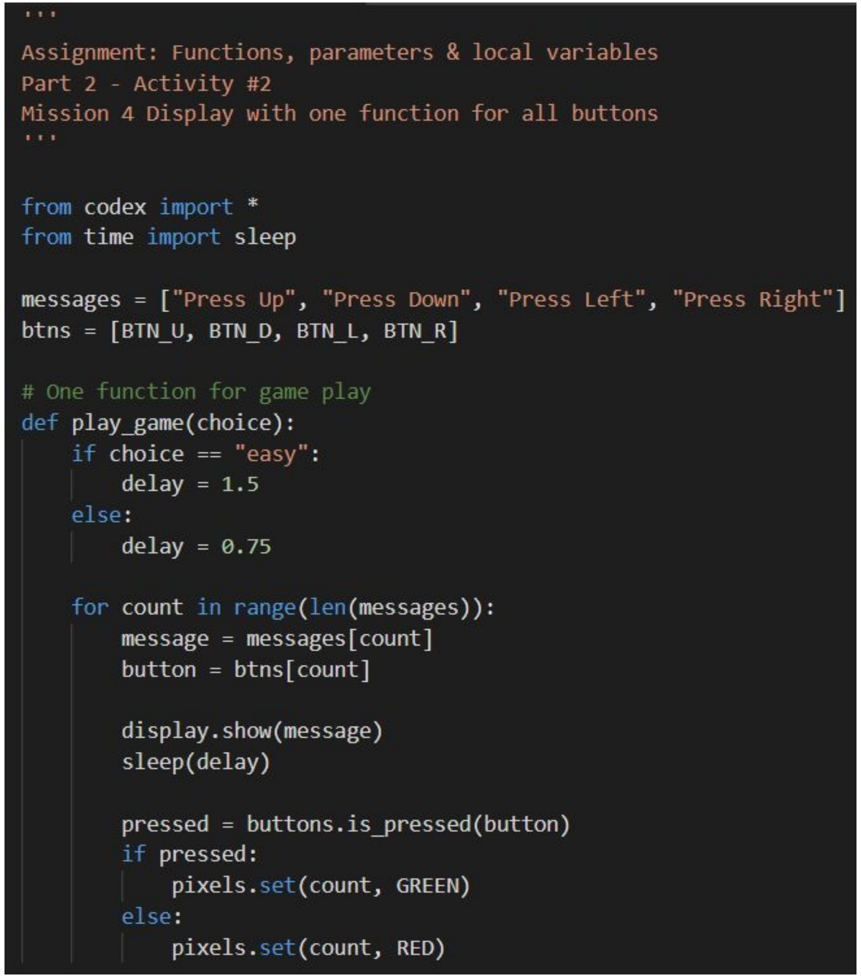


```
delay = random.randrange(1000, 5000)
delay_time = delay / 1000
time.sleep(delay_time)
buttons.was_pressed(BTN_A) #resets early press
pixels.set([GREEN, GREEN, GREEN, GREEN])

start_time = time.ticks_ms()
wait_button()
end_time = time.ticks_ms()
elapsed = time.ticks_diff(end_time, start_time)
display.print("Reaction Time")
display.print(elapsed)
display.print("milliseconds")
time.sleep(3)

play_again()
```

Step #2

Create PT Practice #3

- This program also plays a game
- It will only play one time
- Then you must manually restart to play again
- Use the same concept here to play again without restarting

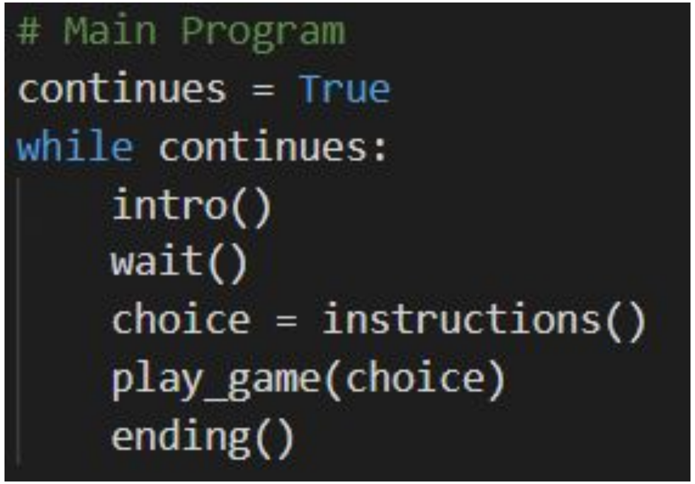


```
'''
Assignment: Functions, parameters & local variables
Part 2 - Activity #2
Mission 4 Display with one function for all buttons
'''

from codex import *
from time import sleep

messages = ["Press Up", "Press Down", "Press Left", "Press Right"]
btns = [BTN_U, BTN_D, BTN_L, BTN_R]

# One function for game play
def play_game(choice):
    if choice == "easy":
        delay = 1.5
    else:
        delay = 0.75

    for count in range(len(messages)):
        message = messages[count]
        button = btns[count]

        display.show(message)
        sleep(delay)

        pressed = buttons.is_pressed(button)
        if pressed:
            pixels.set(count, GREEN)
        else:
            pixels.set(count, RED)
```

Step #2

Create PT Practice #3

- Add a global Boolean variable to the main program
- Add a while loop with the Boolean variable
- Remember to indent all the function calls inside the while loop

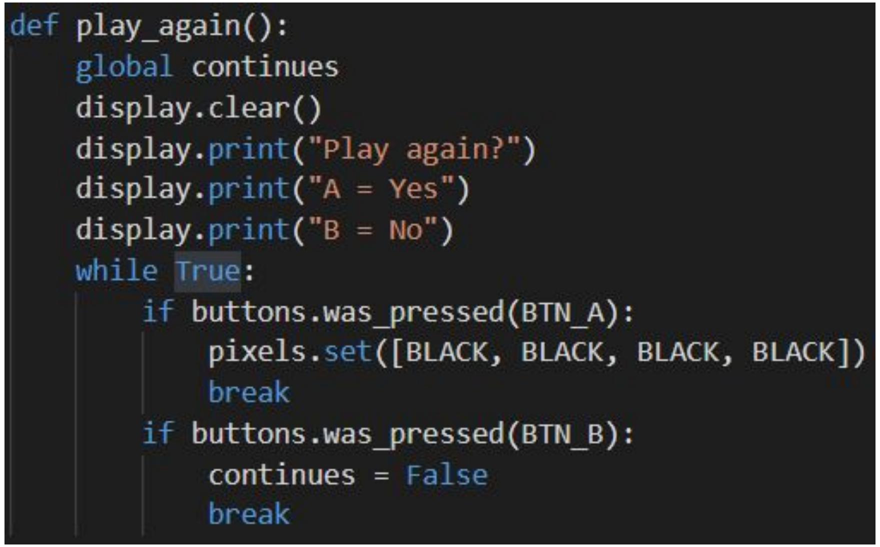


```
# Main Program
continues = True
while continues:
    intro()
    wait()
    choice = instructions()
    play_game(choice)
    ending()
```

Step #2

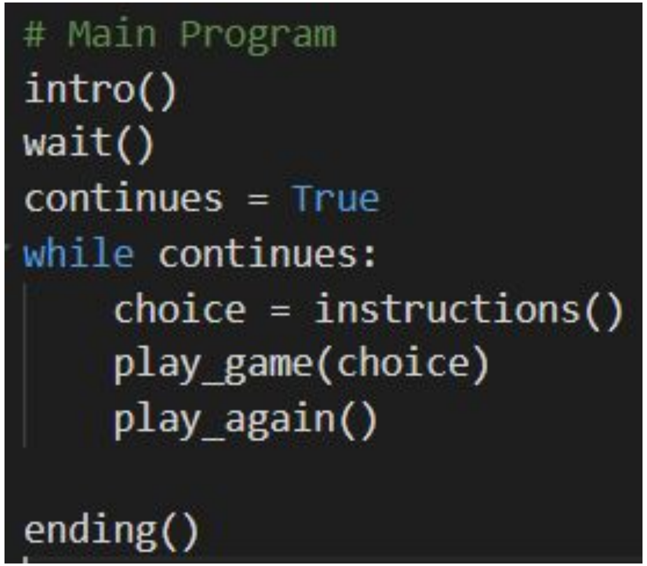
Create PT Practice #3

- Add a function that asks the user to play again
- This can be just like the one you did for Reaction_time
- You can choose to reset the pixels before playing again in BTN_A



```
def play_again():
    global continues
    display.clear()
    display.print("Play again?")
    display.print("A = Yes")
    display.print("B = No")
    while True:
        if buttons.was_pressed(BTN_A):
            pixels.set([BLACK, BLACK, BLACK, BLACK])
            break
        if buttons.was_pressed(BTN_B):
            continues = False
            break
```

Step #2



```
# Main Program
intro()
wait()
continues = True
while continues:
    choice = instructions()
    play_game(choice)
    play_again()

ending()
```

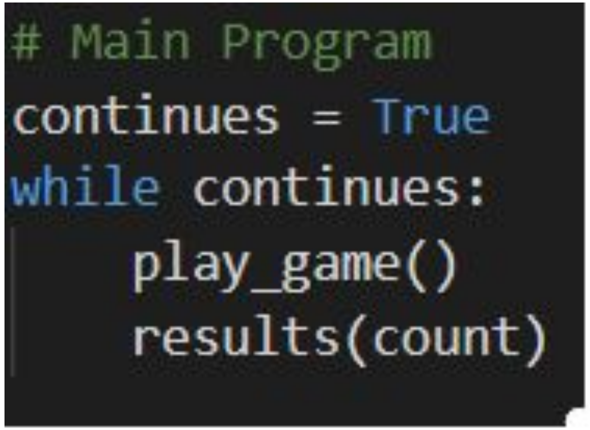
Create PT Practice #3

- Call the function in the main program
- You can decide what functions you want in the loop and what you want outside the loop
- Test and debug – can you play again without restarting?

Step #3

Create PT Practice #4

- This program plays the same game as Practice #3, but with a different ending and no choice
- Make the same changes to this code
- Start by adding a Boolean variable and while loop



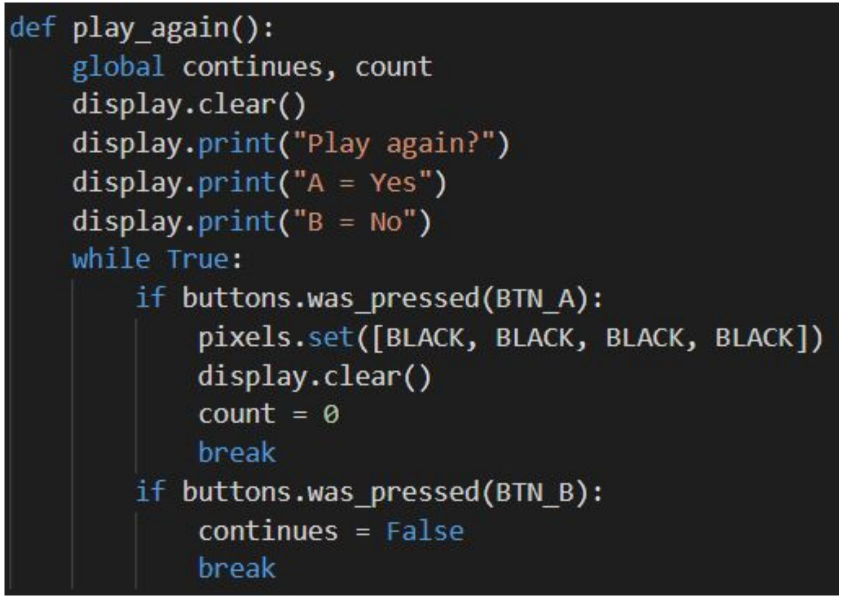
```
# Main Program
continues = True
while continues:
    play_game()
    results(count)
```

The original code had an ending() function. This won't really be the ending anymore, so you could change the function name to results()

Step #3

Create PT Practice #4

- Add a play_again() function
- Think about what you want to happen if you are playing again (BTN_A)
    - All pixels off (BLACK)
    - Screen is cleared
    - Count is reset to 0
    - Count is global, so ….

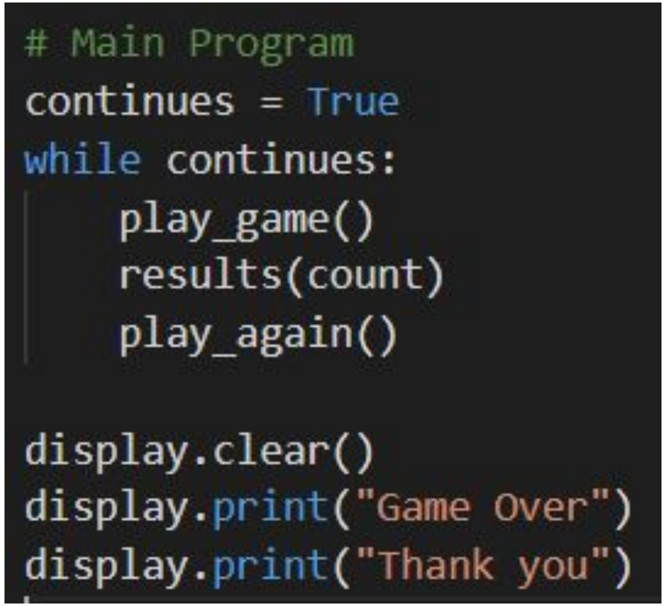


```
def play_again():
    global continues, count
    display.clear()
    display.print("Play again?")
    display.print("A = Yes")
    display.print("B = No")
    while True:
        if buttons.was_pressed(BTN_A):
            pixels.set([BLACK, BLACK, BLACK, BLACK])
            display.clear()
            count = 0
            break
        if buttons.was_pressed(BTN_B):
            continues = False
            break
```

Step #3

Create PT Practice #4

- Call the function in the main program
- You may want to add an actual ending so the user knows the program has ended

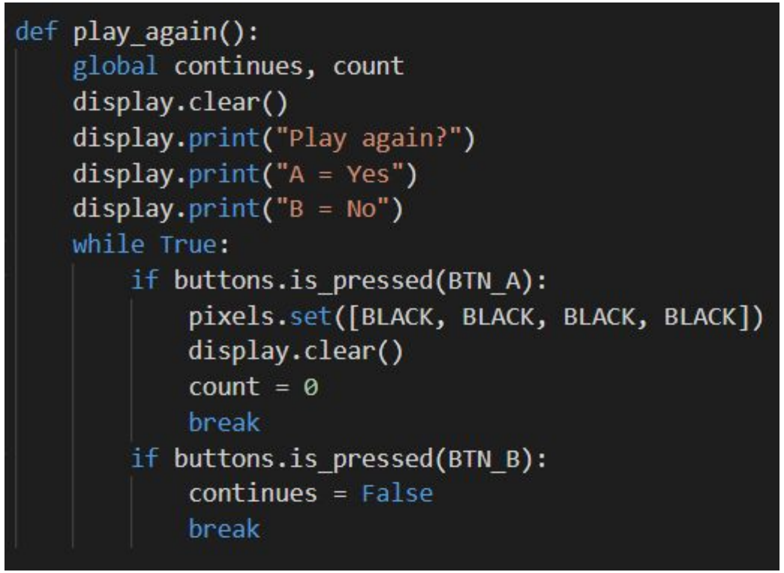


```
# Main Program
continues = True
while continues:
    play_game()
    results(count)
    play_again()

display.clear()
display.print("Game Over")
display.print("Thank you")
```

Step #3

Create PT Practice #4

- Test and debug
- If the play_again() function doesn't seem to work the way you want (not waiting for a button press) try using **buttons.is_pressed** instead
    - This is always an option when a buttons.was_pressed doesn't seem to work correctly.



```python
def play_again():
    global continues, count
    display.clear()
    display.print("Play again?")
    display.print("A = Yes")
    display.print("B = No")
    while True:
        if buttons.is_pressed(BTN_A):
            pixels.set([BLACK, BLACK, BLACK, BLACK])
            display.clear()
            count = 0
            break
        if buttons.is_pressed(BTN_B):
            continues = False
            break
```

Step #4

Try this on your own

- Open another program that plays a game.
    - Mission 4 - Display
    - Display3
    - One of your remix projects
- Add a Boolean variable, loop and function so the game can be played again without restarting.

Congratulations!

By completing this extra practice you can:

- Use a Boolean variable
- Use a while loop with a Boolean variable
- Change the value of the Boolean variable to end the loop
- Use these techniques to play a game until choosing to quit

FIRIA LABS