

# Beginner's Python workshop

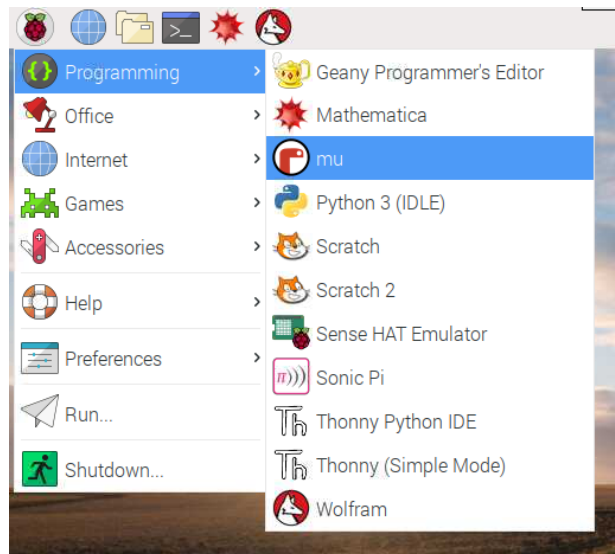
In this 1 hour workshop we will write a simple game in Python using the Jam-supplied Raspberry Pi workstations. Bring your own USB stick if you want to take your finished game away at the end of the workshop. No prior programming experience is needed.

We're going to make a game where the player is a chicken and has to eat as much food as possible within the time limit.

We will write the game using the **mu** editor on the Raspberry Pi.

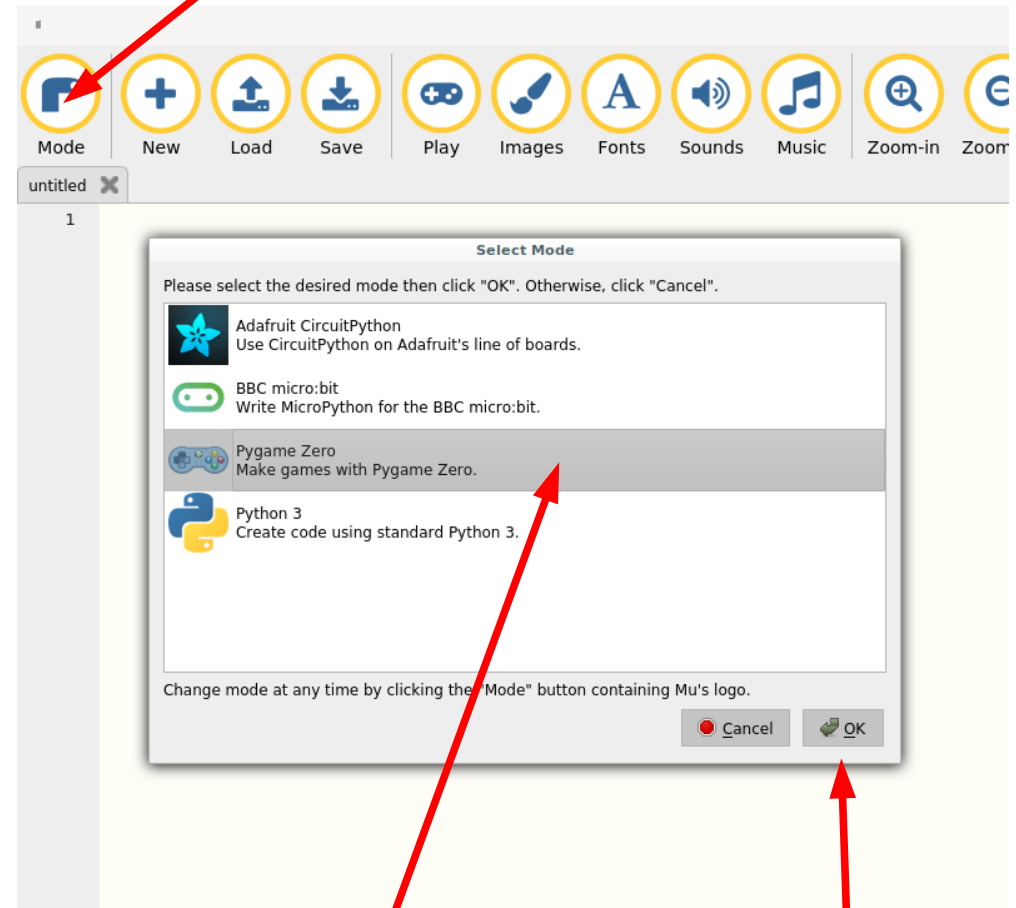
Mu lets us easily write and run games using **Pygame Zero**, and that's what we're going to be using for today's workshop.

Start mu from the programming menu.



Now put mu into the "Pygame Zero" mode.

1. Click "Mode"



2. Select "Pygame Zero"

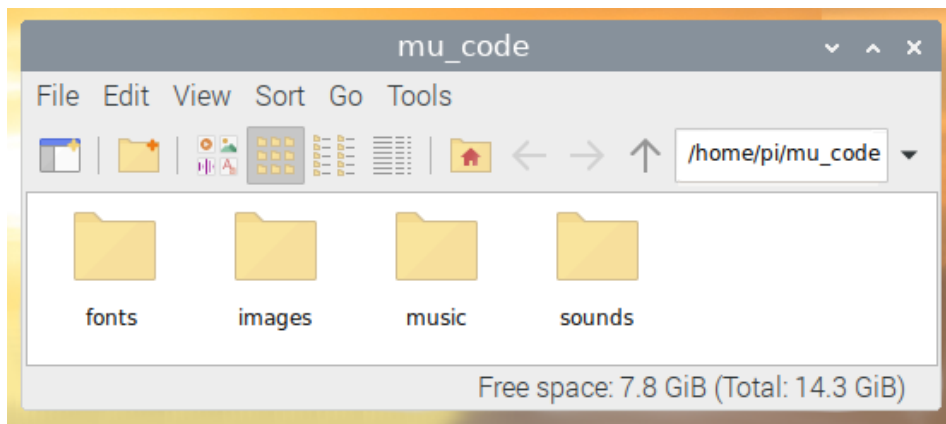
3. Click OK

## Before we begin

Your instructor should have already set up the computers with everything you need to complete this workshop.

You will be entering the Python code yourself, but including some art and sounds that have already been created for you.

Inside your home directory `/home/pi` there should be another directory called `mu_code`. Inside the `mu_code` directory there are directories for the images, sounds, and so on.



We will save the code for our game as a new file within in the `mu_code` directory. Pygame Zero will look for a directory called `images` in the same directory as our code when we run it. If you want to add a new asset to your game you will need to make sure it is placed in the right directory so Pygame Zero can find it.

This booklet will show you a listing of code on each page. Updates to the previous page's code will be shown highlighted in **green**. Usually, only a little bit of code will change from one part to the next. You can leave most of the code as it is each time, you only need to update the highlighted lines.

The mu editor and Pygame Zero work together to take care of a lot of jobs for us behind the scenes, including running a **game loop** for us. A loop contains code which is run again and again, repeatedly. In our game we want to draw the screen, then update what is on the screen, then draw it again, and so on.

Pygame Zero will automatically run code in functions called "draw" and "update" in a loop. We don't need to set this up, it will be done for us as long as we write functions called "draw" and "update".

Run once, at beginning

Code not in  
functions

Run these two  
functions one after  
the other, again and  
again, forever.



draw function

update function

Only run if called

Other functions

## Step 1- Draw a blank screen

In the rest of this booklet, you will see instructions on the left, and Python code to type into mu on the right.

Start by clicking the “New” button in mu to create a new Python file, and then enter into the window:

```
def draw():  
    screen.fill((0, 150, 0))
```

This will define a **function** called “draw”. Note the colon at the end of the first line, and the four spaces at the beginning of the second one.

Now click “Play” to run the code. Mu will kindly save the file for you before running it. Python programs are always called something like `program.py` (The “.py” means it is a python file). Enter the name “mygame.py”, make sure you are saving in the “/home/pi/mu\_code” directory, and click “Save”. The code will now run, and you will see a blank green screen.

(Remember “draw” is a special function in Pygame Zero. Code in the “draw” function will be run every time the game needs to redraw the screen, which will be done for us constantly behind the scenes.)

Code starting with a # in Python is called a **comment** and it doesn’t get run. You can put whatever you want after the # character. It’s useful for making notes in your program so you understand it when you come back to it.

```
def draw():  
    # Fill the screen with a green colour.  
    # This will be like a field for the player  
    # to walk about in.  
    #  
    # The numbers in brackets represent the level  
    # of red, green, and blue in the fill colour.  
    # The range is between 0 and 255.  
    #  
    # If we did screen.fill((0, 255, 0)) instead,  
    # then the green colour would be really bright,  
    # so to make things less distracting we’ve  
    # chosen a lower value.  
    #  
    # Can you make the screen red instead?  
    # What about yellow?  
    #  
    # Oh, by the way, you don’t need to read these  
    # comments if you don’t want, they are just  
    # here to tell you extra bits and pieces you  
    # might want to know! Don’t get hung up on  
    # them, you can safely skip them and come  
    # back later if you want.  
    screen.fill((0, 150, 0))
```

## Step 2 – Add a chicken

Before you do anything else you'll need to stop the game so we can make some changes to it. The button that you clicked to "Play" the game should have changed to say "Stop". Click on "Stop" so you can go and make some changes to your code. In future we won't mention this, so try and remember to click "Stop" when you want to edit your code.

Pygame Zero lets us load images and sounds to use in our game. We've already downloaded some sounds and images to use in our game, so we can just start adding them.

Our player is going to be a chicken, so we'll create an **Actor** using the chicken image, and assign it to the variable `player`.

Actors are Pygame Zero's way of representing the players, items, enemies and so on that will inhabit our game world. We'll be using actors in this game for the chicken and for the food it will eat.

Now whenever we want to refer to the chicken in our program, we can just type "`player`" instead.

Update your code to create the player, and then "Play" the updated code to see the chicken on the screen.

```
player = Actor('chicken')
# The screen in Pygame Zero is made up of little
# squares called "pixels". You can think of this
# like a big piece of squared paper. It is 800
# squares wide and 600 squares tall.
#
# When we tell Pygame Zero a position on the
# screen, we first tell it the number of pixels
# from the left (sometimes called the "x
# coordinate") followed by the number of pixels
# down from the top of the screen ("the y
# coordinate").
#
# Here, we are telling Pygame Zero that we want
# the centre of the chicken to be 400 pixels from
# the left hand side of the screen, and 400 pixels
# down from the top of the screen.
player.center = 400, 400

def draw():
    screen.fill((0, 150, 0))
    player.draw()
```



### Step 3 – Make the chicken controllable

This wouldn't be much of a game if we couldn't control the chicken! We want to use the arrow keys to move the chicken around the field.

We're going to use another special Pygame Zero function called "update". Like "draw", this gets run regularly in the background, and lets us make changes to the state of the game based on what the actors and the player are doing.

All we're going to do right now in the "update" function is call another new function, called "check\_keys".

"check\_keys" has the job of looking to see if the player is pressing any of the arrow keys, and then moving the player a little bit in the direction they are pressing.

You can see in `check_keys` that we are now talking about `player.x` and `player.y`. These are numeric values that describe the distance of the player from the left hand side of the screen (`player.x`) and from the top of the screen (`player.y`). We want to make sure the player is not allowed to be too near to any of the edges of the screen, so we check these x and y values to make sure they are inside the area we are going to draw. We only let the player move in a particular direction if they are not too close to that edge.

```
player = Actor('chicken')
player.center = 400, 400
```

```
def draw():
    screen.fill((0, 150, 0))
    player.draw()
```

```
def update():
    # We could have just written all of the new
    # code here in the 'update' function, but I
    # like it more when we split it up like this.
    #
    # Later we will add more code to 'update'.
    #
    # By separating out 'check_keys' we make it
    # more obvious what each part of the program
    # is doing.
    check_keys()
```

```
def check_keys():
    # The 'if' statement is a really important
    # part of Python! Any time you want to run
    # a piece of code sometimes, but not all the
    # time, you probably need an 'if' statement!
    if keyboard.left:
        if player.x > 40: player.x -= 5
    if keyboard.right:
        if player.x < 760: player.x += 5
    if keyboard.up:
        if player.y > 50: player.y -= 5
    if keyboard.down:
        if player.y < 550: player.y += 5
```

## Step 4 – Add some food for the chicken

To give the chicken something to do, we will create some food. I think chickens would probably like to eat pears, so I'm going to give it a pear.

The pear will be an `Actor`, just like the chicken. We'll put it over on the right hand side of the screen for now.

We want the chicken to be able to eat the food.

Pygame Zero gives us the function `collidepoint` which lets us work out if any part of an `Actor` is touching another point. So we can check if the chicken is colliding with the food by saying

```
if player.collidepoint(food.pos):  
    do_something()
```

(`food.pos` is a variable representing the position of the food).

Now, we're using a little trick here. If the chicken is touching the food, we want it to look like the food has been eaten. But what we're actually doing is moving the centre of the food to -100, -100, which is off the screen, so we can't see it any more!

```
player = Actor('chicken')  
player.center = 400, 400  
food = Actor('pear')  
food.center = 700, 500  
  
def draw():  
    screen.fill((0, 150, 0))  
    player.draw()  
    food.draw()  
  
def update():  
    check_keys()  
    if player.collidepoint(food.pos):  
        food.center = -100, -100  
  
def check_keys():  
    global player  
    if keyboard.left:  
        if player.x > 40: player.x -= 5  
    if keyboard.right:  
        if player.x < 760: player.x += 5  
    if keyboard.up:  
        if player.y > 50: player.y -= 5  
    if keyboard.down:  
        if player.y < 550: player.y += 5
```

## Step 5 – Put the food in a random place

It would be a bit boring if the food was always in the same place. So we're going to make the food appear somewhere unexpected.

To do this, we will use a Python library called `random`. Libraries are big pieces of pre-written Python code that we can include in our programs. Often you will find someone has already written a library that does something you want, so you can just include that library in your program and make use of the functionality it provides.

The `random` library lets us use the function `randint` to pick a number between two values.

Just like when we were moving the chicken and didn't want it to get too close to the edges, we will set the fruit in a random position that is somewhere between the edges of the screen, but not too close to any of them.

Remember, when we are setting a position, the first number is the x coordinate (distance from left) and the second number is the y coordinate (distance from top).

We won't keep including the `check_keys` function from now on in our listing because it isn't going to change. Just leave it like it was from last time.



```
import random
```

```
player = Actor('chicken')  
player.center = 400, 400  
food = Actor('pear')
```

```
# The brackets at the start and end here create  
# a 'tuple', as well as allowing the expression  
# to span two lines.
```

```
food.center = (random.randint(60, 740),  
               random.randint(60, 540))
```

```
def draw():  
    screen.fill((0, 150, 0))  
    player.draw()  
    food.draw()
```

```
def update():  
    check_keys()  
    if player.collidepoint(food.pos):  
        food.center = -100, -100
```

```
def check_keys():  
    ...
```

## Step 6 – Make lots of food

It's time for some more trickery!

We want the chicken to eat lots of pears, but we've already seen how we can trick the player into thinking they ate a pear just by moving it. What if every time the chicken "eats" a pear, that pear just moves somewhere else on the screen? It will look like the chicken ate the pear, and now there is a new pear somewhere else for it to go and eat!

When we first added the pear in part 5, we placed it in a random location. We will take that code and place it in its own function called `move_food`. Then we can call the `move_food` function whenever we want to put the pear in a new random location.

We want to do that once at the very start of the game, so the food doesn't always start in the same place.

Then we want to do it once every time the chicken collides with the food, so it moves somewhere else on screen. This will replace our previous trick of moving the food off the screen when it is eaten.

```
import random

player = Actor('chicken')
player.center = 400, 400
food = Actor('pear')

# Remember, functions like this don't get run
# until we call them.
def move_food():
    food.center = (random.randint(60, 740),
                  random.randint(60, 540))

# Call the move_food function. Don't miss out this
# line, or your first pear will always be in the
# top left of the screen!
move_food()

def draw():
    screen.fill((0, 150, 0))
    player.draw()
    food.draw()

def update():
    check_keys()
    if player.collidepoint(food.pos):
        move_food()

def check_keys():
    ...
```



## Step 7 – Add a score

To keep track of the player's progress, we'll give them a score for how much fruit they have eaten.

To do this, we add a variable called "score" which will start at 0 (because the player hasn't eaten any food yet).

When the chicken eats a pear, we'll award the player 5 points by adding 5 to the score, right before we move the food.

When we draw the screen, we'll also write the score in the top left of the screen. The line:

```
screen.draw.text("Score: " + str(score), (20, 20))
```

will write some text (the first argument) to a particular place on the screen (the second argument). In Python, whole numbers are called **integers** and sequences of characters (text) are called `strings`.

We can't add an integer directly to a string because we can only add variables of the same type. So we convert the number `score` into a string using the `str` function before adding it to the text "Score: ". The end result will be a string saying something like "Score: 5".

```
import random

player = Actor('chicken')
player.center = 400, 400
food = Actor('pear')
score = 0

def move_food():
    food.center = (random.randint(60, 740),
                  random.randint(60, 540))

move_food()

def draw():
    screen.fill((0, 150, 0))
    player.draw()
    food.draw()
    screen.draw.text("Score: " + str(score),
                    (20, 20))

def update():
    # The use of 'global' here is kind
    # of an advanced topic, so ask later
    # if you want to know why we use it here!
    global score
    check_keys()
    if player.collidepoint(food.pos):
        score = score + 5
        move_food()

def check_keys():
    ...
```

## Step 8 – Add an ending

To make it into more of a game, lets set a time limit for eating the fruit. Who can eat the most fruit within the allowed time?!

To keep track of whether or not the game has finished, we will add a new variable called “playing”. This will contain either the value `True` or `False`. We call variables like these **boolean** variables.

Any time we want to see if the game has finished or not, we can do a check like:

```
if playing:
    do stuff to do with the game play
else:
    do "Game over" type stuff
```

We will use the built in `clock` feature of Pygame Zero to set a countdown timer running in the background when the game begins. When the countdown timer runs out, the `game_over` function will be called, which will set the value of `playing` to `False`.

When the game is over, we'll display the final score in big letters so we can show the person next to us how awesome we are at eating pears.

```
import random

player = Actor('chicken')
player.center = 400, 400
food = Actor('pear')
score = 0
playing = True

def move_food():
    food.center = (random.randint(60, 740),
                  random.randint(60, 540))

move_food()

def draw():
    screen.fill((0, 150, 0))
    if playing:
        player.draw()
        food.draw()
        screen.draw.text("Score: " + str(score), (20, 20))
    else:
        screen.draw.text("Time up! You scored " + str(score),
                        center=(400, 500), fontsize=60)

def update():
    global score
    if playing:
        check_keys()
        if player.collidepoint(food.pos):
            score = score + 5
            move_food()

def check_keys():
    ...

def game_over():
    global playing
    playing = False

# The number 20 here makes the game last 20 seconds
clock.schedule(game_over, 20)
```

## Step 9 – Advanced! Add another kind of food

To add a bit of variety, we will give the chicken another special type of food to eat. This food will appear more rarely, and will be worth more points.

We'll use all the tricks we learned so far to add a treat into the mix - a pineapple!

Instead of moving the treat when the chicken eats it, we're going to hide it off screen, so the player can't just keep eating treats and receive loads of points. Then every few seconds we want it to appear again somewhere else on the screen.

We'll use another feature of Pygame Zero's `clock` to move the treat once every few seconds. If we tell `clock` to `schedule_interval` instead of `schedule` then it will keep performing the action again and again, which is what we want here. The treat will jump to a new location every few seconds, which will also have the effect of bringing it back on screen if it had previously been hidden.



```
import random

player = Actor('chicken')
player.center = 400, 400
food = Actor('pear')
treat = Actor('pineapple')
score = 0
playing = True

def move_food():
    food.center = (random.randint(60, 740), random.randint(60, 540))

def move_treat():
    treat.center = (random.randint(60, 740), random.randint(60, 540))

def hide_treat():
    treat.center = -100, -100

move_food()
hide_treat()

def draw():
    screen.fill((0, 150, 0))
    if playing:
        player.draw()
        food.draw()
        treat.draw()
        screen.draw.text("Score: " + str(score), (20, 20))
    else:
        screen.draw.text("Time up! You scored " + str(score),
                        center=(400, 500), fontsize=60)

def update():
    global score
    if playing:
        check_keys()
        if player.collidepoint(food.pos):
            score = score + 5
            move_food()
        if player.collidepoint(treat.pos):
            score = score + 10
            hide_treat()

def check_keys():
    ...

def game_over():
    global playing
    playing = False

clock.schedule_interval(move_treat, 3)
clock.schedule(game_over, 20)
```

## Step 10 – Add a picture as a background

Rather than the green field, lets add a nice picture as a background for the chicken to walk around.

`field` is the name of an image file (like the chicken, pear, and pineapple).

```
import random

player = Actor('chicken')
player.center = 400, 400
food = Actor('pear')
treat = Actor('pineapple')
score = 0
playing = True

def move_food():
    food.center = (random.randint(60, 740), random.randint(60, 540))

def move_treat():
    treat.center = (random.randint(60, 740), random.randint(60, 540))

def hide_treat():
    treat.center = -100, -100

move_food()
hide_treat()

def draw():
    # Note that this next line has *replaced* the screen.fill()
    # which we had in here previously!
    screen.blit('field', (0, 0))
    if playing:
        player.draw()
        food.draw()
        treat.draw()
        screen.draw.text("Score: " + str(score), (20, 20))
    else:
        screen.draw.text("Time up! You scored " + str(score),
                        center=(400, 500), fontsize=60)

def update():
    ...

def check_keys():
    ...

def game_over():
    global playing
    playing = False

clock.schedule_interval(move_treat, 3)
clock.schedule(game_over, 20)
```

## Step 11 – Add sounds and music

Most games have sounds and music, and it's easy for us to add these features using Pygame Zero's built in support.

`cluck` and `shout` are the names of sound files.

"backing" is the name of a music file. If it doesn't work for you, try the alternative file "ogg\_backing" to load music in a different format.



**Congratulations! Game complete!**

See overleaf for the full game listing.

All of the files for this workshop are available to download online:

<https://github.com/rbricheno/pygamezero-workshop>

Thanks for coming, we hope you had fun :-)

```
import random

player = Actor('chicken')
player.center = 400, 400
food = Actor('pear')
treat = Actor('pineapple')
score = 0
playing = True

def move_food():
    food.center = (random.randint(60, 740), random.randint(60, 540))

def move_treat():
    treat.center = (random.randint(60, 740), random.randint(60, 540))

def hide_treat():
    treat.center = -100, -100

move_food()
hide_treat()

def draw():
    ...
def update():
    global score
    if playing:
        check_keys()
        if player.collidepoint(food.pos):
            score = score + 5
            sounds.cluck.play()
            move_food()
        if player.collidepoint(treat.pos):
            score = score + 10
            sounds.shout.play()
            hide_treat()

def check_keys():
    ...

def game_over():
    global playing
    playing = False

clock.schedule_interval(move_treat, 3)
clock.schedule(game_over, 20)

# Wow that was much too loud!
music.set_volume(0.2)
music.play("backing")
```



The listing for the finished game is as follows:

```
import random

player = Actor('chicken')
player.center = 400, 400
food = Actor('pear')
treat = Actor('pineapple')
score = 0
playing = True

def move_food():
    food.center = (random.randint(60, 740),
                  random.randint(60, 540))

def move_treat():
    treat.center = (random.randint(60, 740),
                   random.randint(60, 540))

def hide_treat():
    treat.center = -100, -100

move_food()
hide_treat()

def draw():
    screen.blit('field', (0, 0))
    if playing:
        player.draw()
        food.draw()
        treat.draw()
        screen.draw.text("Score: " + str(score), (20, 20))
    else:
        screen.draw.text("Time up! You scored " + str(score),
                        center=(400, 500), fontsize=60)
```

```
def update():
    global score
    if playing:
        check_keys()
        if player.collidepoint(food.pos):
            score = score + 5
            sounds.cluck.play()
            move_food()
        if player.collidepoint(treat.pos):
            score = score + 10
            sounds.shout.play()
            hide_treat()

def check_keys():
    global player
    if keyboard.left:
        if player.x > 40: player.x -= 5
    if keyboard.right:
        if player.x < 760: player.x += 5
    if keyboard.up:
        if player.y > 50: player.y -= 5
    if keyboard.down:
        if player.y < 550: player.y += 5

def game_over():
    global playing
    playing = False

clock.schedule_interval(move_treat, 3)
clock.schedule(game_over, 20)
music.set_volume(0.2)
music.play("backing")
```