PROJECT – Flappy Bird

Virtual Environment:

If we use different libraries, there would be confusion. Therefore, we need to create a container in which we install these versions. This container is called ***virtual environment***.

PIP → is a package manager, through which we can install an library that we need.

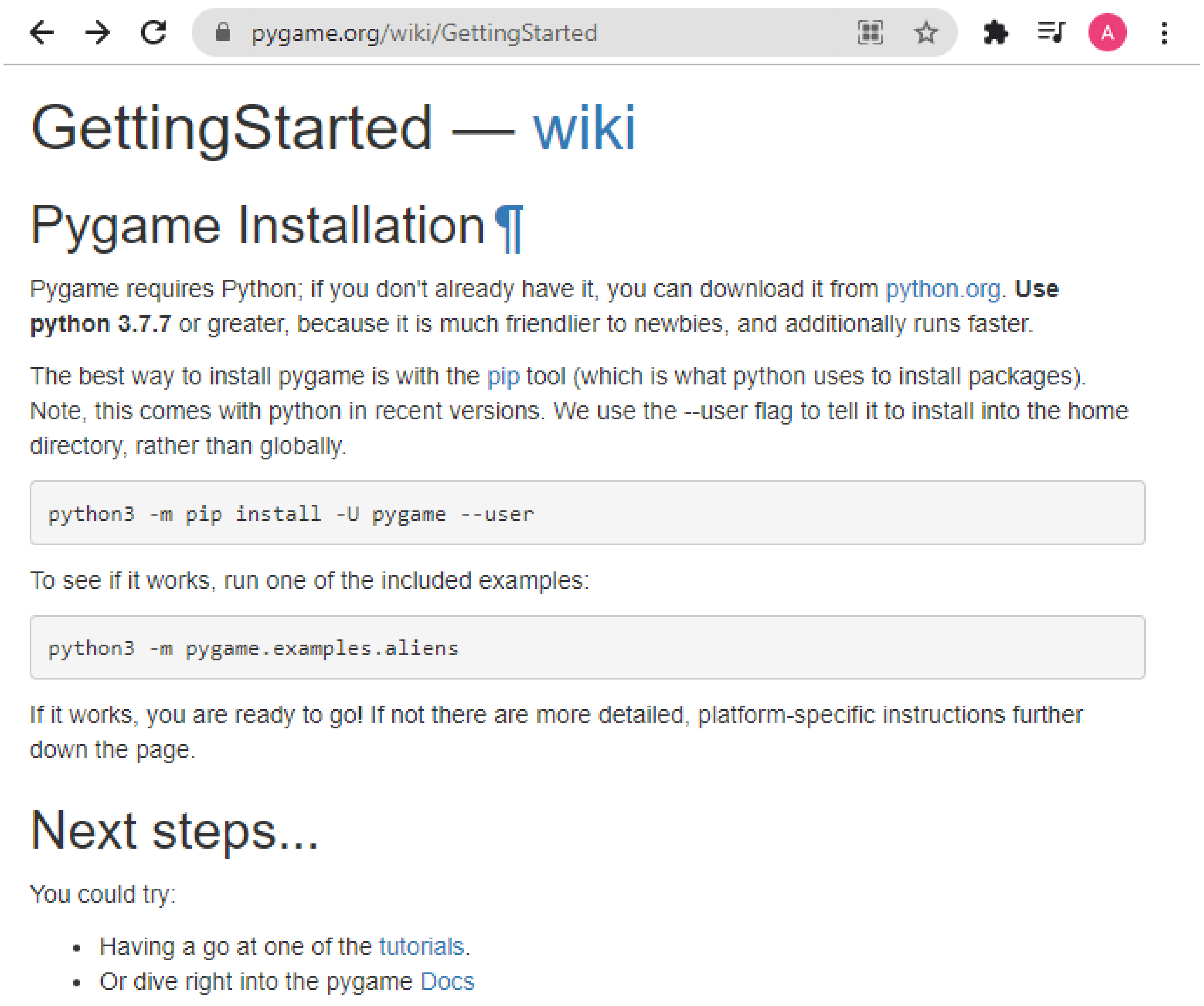
python -n venv .venv → this is a *syntax* to create a virtual environment. It will create a virtual directory '**.venv**'. Inside this library, all the project libraries that we will be using will be installed.

We will create a '🐍 main.py' file in the same folder. Then we will initiate git and do the first commit as "***adding bare main.py***".

We will create another file, '**requirements.txt**' file. This file will have all the libraries that we will use for this project. If you don't have pygame library already installed in your libraries, then you can use, '**python3 -m pip install -U pygame**'. Use "***Python Pipy***" to get the latest version.

In the README.md file, we can write all the steps we take in this file. Now let's commit the '**requirements.txt**' file to our git account as "***adding pygame to requirments.txt***" and then we will commit, '**README.md**' as "***adding steps to install library***".

We will go to https://www.pygame.org/wiki/GettingStarted to get details about how to get started.



GettingStarted — wiki

Pygame Installation ¶

Pygame requires Python; if you don't already have it, you can download it from python.org. **Use python 3.7.7** or greater, because it is much friendlier to newbies, and additionally runs faster.

The best way to install pygame is with the pip tool (which is what python uses to install packages). Note, this comes with python in recent versions. We use the --user flag to tell it to install into the home directory, rather than globally.

```
python3 -m pip install -U pygame --user
```

To see if it works, run one of the included examples:

```
python3 -m pygame.examples.aliens
```

If it works, you are ready to go! If not there are more detailed, platform-specific instructions further down the page.

Next steps...

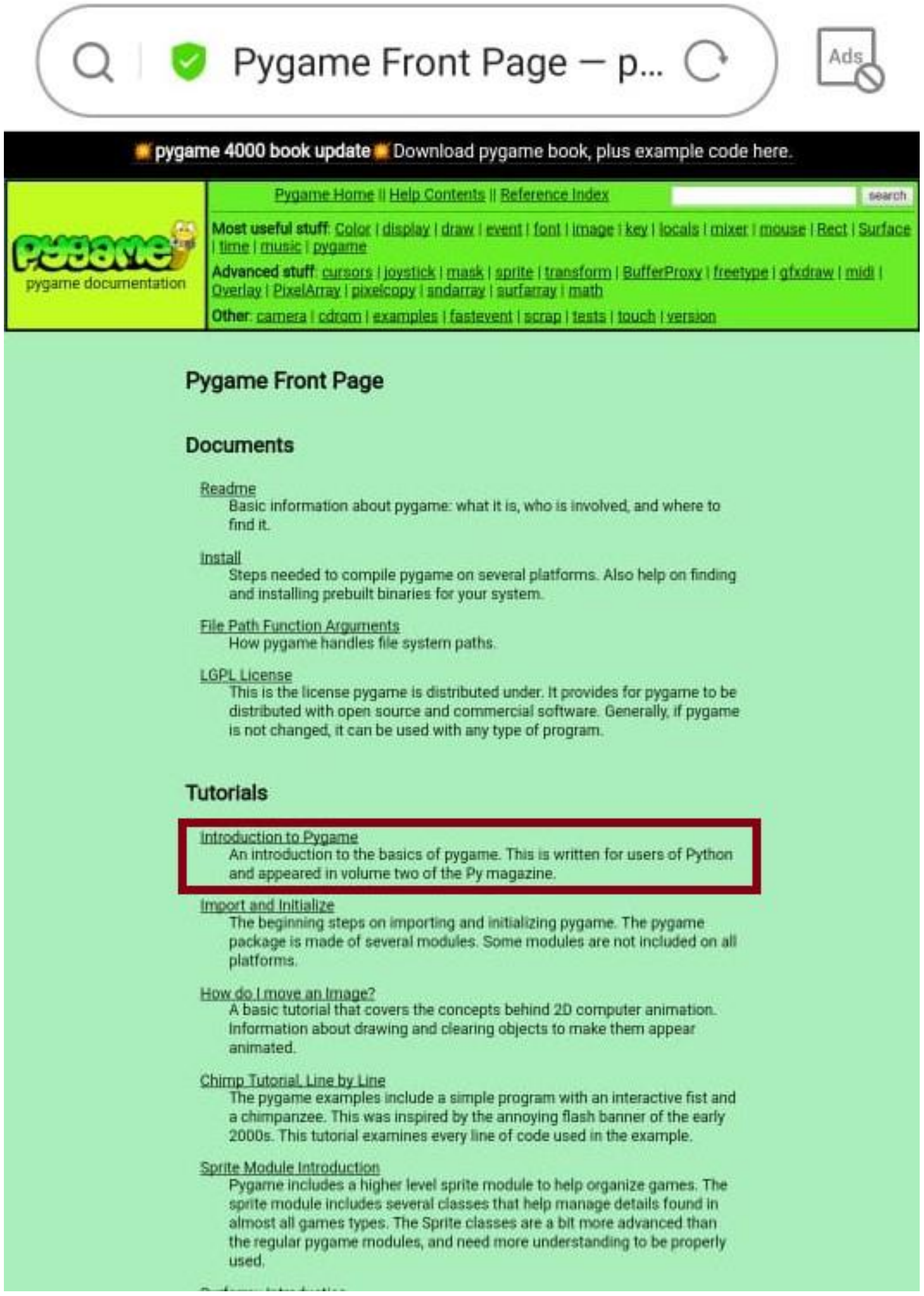
You could try:

- Having a go at one of the tutorials.
- Or dive right into the pygame Docs

This page gives us links to tutorials and Docs of how to get started. See the "Next steps…"

Click on the links to go to the Docs page.

Once we click the Docs link, the following page will open.


Pygame Front Page — p...

pygame 4000 book update Download pygame book, plus example code here.

pygame documentation

Pygame Home || Help Contents || Reference Index

Most useful stuff: Color | display | draw | event | font | image | key | locals | mixer | mouse | Rect | Surface | time | music | pygame

Advanced stuff: cursors | joystick | mask | sprite | transform | BufferProxy | freetype | gfxdraw | midi | Overlay | PixelArray | pixelcopy | sndarray | surfarray | math

Other: camera | cdrom | examples | fastevent | scrap | tests | touch | version

Pygame Front Page

Documents

Readme
Basic information about pygame: what it is, who is involved, and where to find it.

Install
Steps needed to compile pygame on several platforms. Also help on finding and installing prebuilt binaries for your system.

File Path Function Arguments
How pygame handles file system paths.

LGPL License
This is the license pygame is distributed under. It provides for pygame to be distributed with open source and commercial software. Generally, if pygame is not changed, it can be used with any type of program.

Tutorials

Introduction to Pygame
An introduction to the basics of pygame. This is written for users of Python and appeared in volume two of the Py magazine.

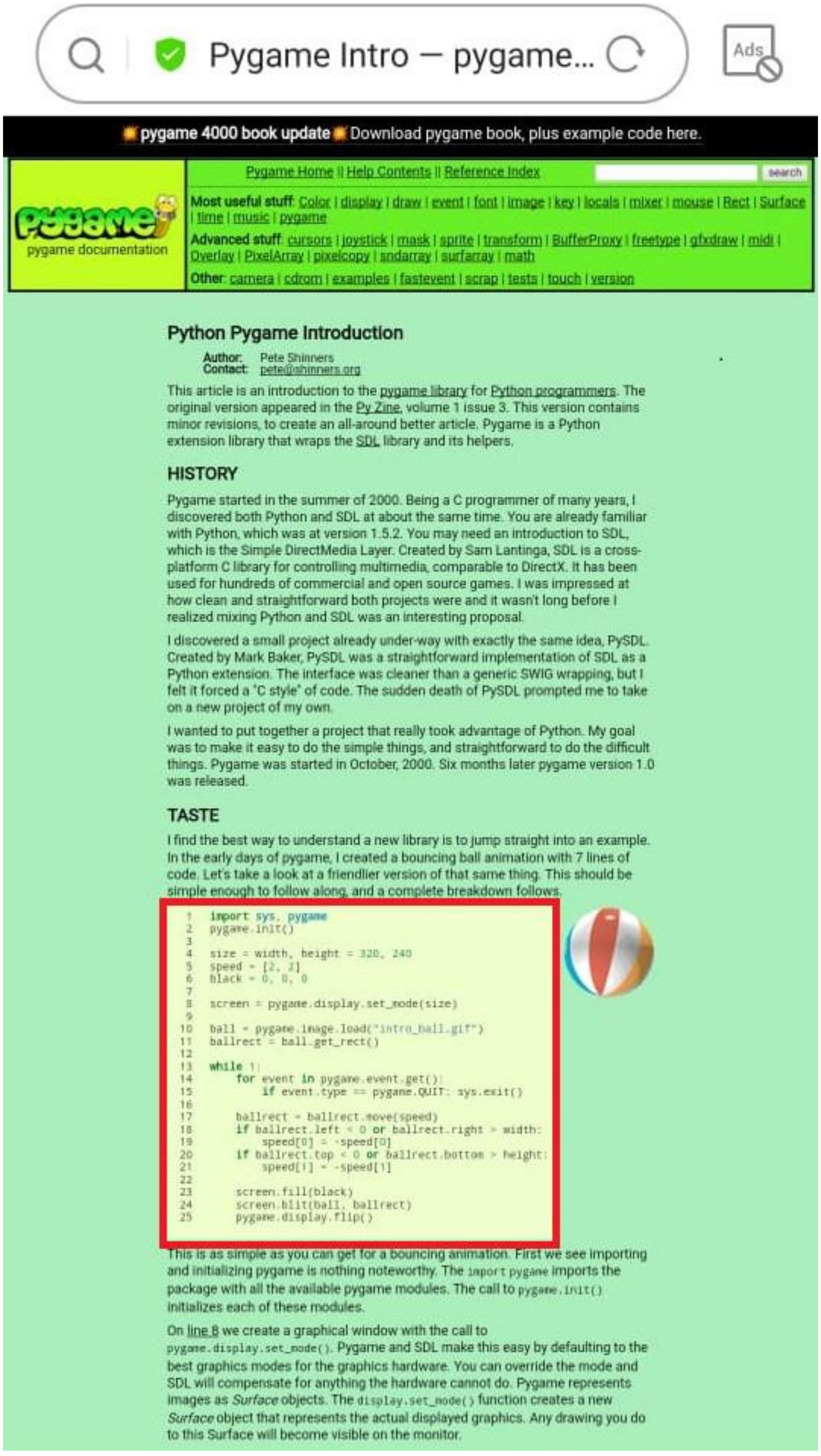
Import and Initialize
The beginning steps on importing and initializing pygame. The pygame package is made of several modules. Some modules are not included on all platforms.

How do I move an Image?
A basic tutorial that covers the concepts behind 2D computer animation. Information about drawing and clearing objects to make them appear animated.

Chimp Tutorial, Line by Line
The pygame examples include a simple program with an interactive fist and a chimpanzee. This was inspired by the annoying flash banner of the early 2000s. This tutorial examines every line of code used in the example.

Sprite Module Introduction
Pygame includes a higher level sprite module to help organize games. The sprite module includes several classes that help manage details found in almost all games types. The Sprite classes are a bit more advanced than the regular pygame modules, and need more understanding to be properly used.

Click on the “***Introduction to Pygame***”.



Pygame Intro — pygame...
Ads

pygame documentation
Pygame Home || Help Contents || Reference Index
search
Most useful stuff: Color | display | draw | event | font | image | key | locals | mixer | mouse | Rect | Surface | time | music | pygame
Advanced stuff: cursors | joystick | mask | sprite | transform | BufferProxy | freetype | gfxdraw | midi | Overlay | PixelArray | pixelcopy | sndarray | surfarray | math
Other: camera | cdrom | examples | fastevent | scrap | tests | touch | version

Python Pygame Introduction

Author: Pete Shinners
Contact: pete@shinners.org

This article is an introduction to the pygame library for Python programmers. The original version appeared in the Py Zine, volume 1 issue 3. This version contains minor revisions, to create an all-around better article. Pygame is a Python extension library that wraps the SDL library and its helpers.

HISTORY

Pygame started in the summer of 2000. Being a C programmer of many years, I discovered both Python and SDL at about the same time. You are already familiar with Python, which was at version 1.5.2. You may need an introduction to SDL, which is the Simple DirectMedia Layer. Created by Sam Lantinga, SDL is a cross-platform C library for controlling multimedia, comparable to DirectX. It has been used for hundreds of commercial and open source games. I was impressed at how clean and straightforward both projects were and it wasn't long before I realized mixing Python and SDL was an interesting proposal.

I discovered a small project already under-way with exactly the same idea, PySDL. Created by Mark Baker, PySDL was a straightforward implementation of SDL as a Python extension. The interface was cleaner than a generic SWIG wrapping, but I felt it forced a "C style" of code. The sudden death of PySDL prompted me to take on a new project of my own.

I wanted to put together a project that really took advantage of Python. My goal was to make it easy to do the simple things, and straightforward to do the difficult things. Pygame was started in October, 2000. Six months later pygame version 1.0 was released.

TASTE

I find the best way to understand a new library is to jump straight into an example. In the early days of pygame, I created a bouncing ball animation with 7 lines of code. Let's take a look at a friendlier version of that same thing. This should be simple enough to follow along, and a complete breakdown follows.

```
1   import sys, pygame
2   pygame.init()
3
4   size = width, height = 320, 240
5   speed = [2, 2]
6   black = 0, 0, 0
7
8   screen = pygame.display.set_mode(size)
9
10  ball = pygame.image.load("intro_ball.gif")
11  ballrect = ball.get_rect()
12
13  while 1:
14      for event in pygame.event.get():
15          if event.type == pygame.QUIT: sys.exit()
16
17      ballrect = ballrect.move(speed)
18      if ballrect.left < 0 or ballrect.right > width:
19          speed[0] = -speed[0]
20      if ballrect.top < 0 or ballrect.bottom > height:
21          speed[1] = -speed[1]
22
23      screen.fill(black)
24      screen.blit(ball, ballrect)
25      pygame.display.flip()
```

This is as simple as you can get for a bouncing animation. First we see importing and initializing pygame is nothing noteworthy. The `import pygame` imports the package with all the available pygame modules. The call to `pygame.init()` initializes each of these modules.

On line 8 we create a graphical window with the call to `pygame.display.set_mode()`. Pygame and SDL make this easy by defaulting to the best graphics modes for the graphics hardware. You can override the mode and SDL will compensate for anything the hardware cannot do. Pygame represents images as *Surface* objects. The `display.set_mode()` function creates a new *Surface* object that represents the actual displayed graphics. Any drawing you do to this Surface will become visible on the monitor.

Copy the code in the white box and we will make changes to it.

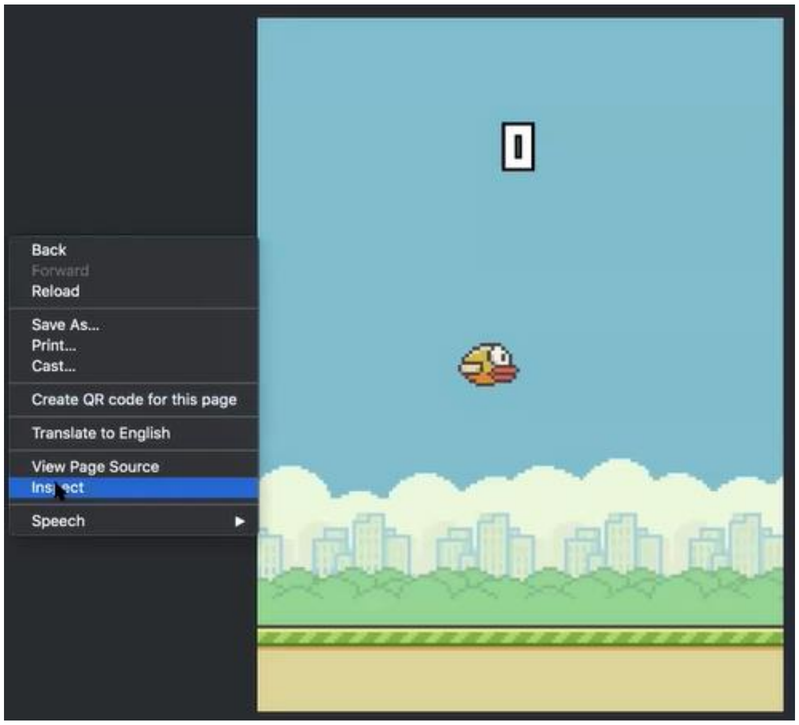
```python
import sys, pygame
pygame.init()

size = width, height = 320, 240
speed = [2, 2]
black = 0, 0, 0

screen = pygame.display.set_mode(size)

ball = pygame.image.load("intro_ball.gif")
ballrect = ball.get_rect()

while 1:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: sys.exit()

    ballrect = ballrect.move(speed)
    if ballrect.left < 0 or ballrect.right > width:
        speed[0] = -speed[0]
    if ballrect.top < 0 or ballrect.bottom > height:
        speed[1] = -speed[1]

    screen.fill(black)
    screen.blit(ball, ballrect)
    pygame.display.flip()
```

We will go through the code and understand what the code is doing.

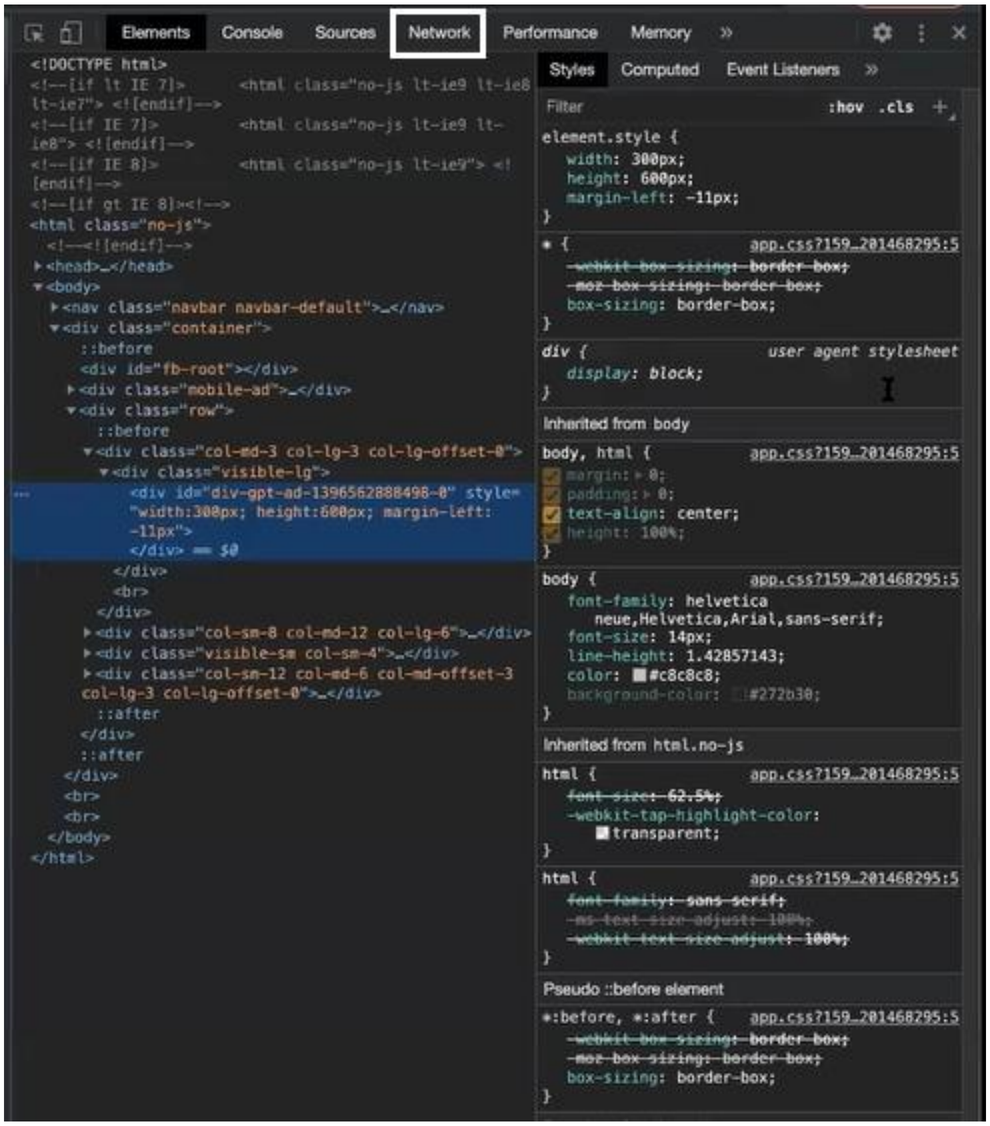
We can clean the code as well at the same time.

```
import sys
import pygame

pygame.init()

WIDTH = 320
HEIGHT = 240

black = 0, 0, 0 # rgb of black is 0, 0, 0
# we can change the color

# to get a screen we use the following command
# we need to have a tuple here
screen = pygame.display.set_mode((WIDTH, HEIGHT))

while True: # this is an infinite loop
    # all the events are in the event
    for event in pygame.event.get():
        # if somebody say quit, we will quit the game
        if event.type == pygame.QUIT: sys.exit()


    screen.fill(black)
    # flip - at the end, it refreshes the game (FPS -
 frames per second)
    # images are very moving very fast
    pygame.display.flip()
```

Let’s open the game in google chrome and then we can get details about the game. On the page, right-click and go to ‘**Inspect**’ which will open the developer’s page.



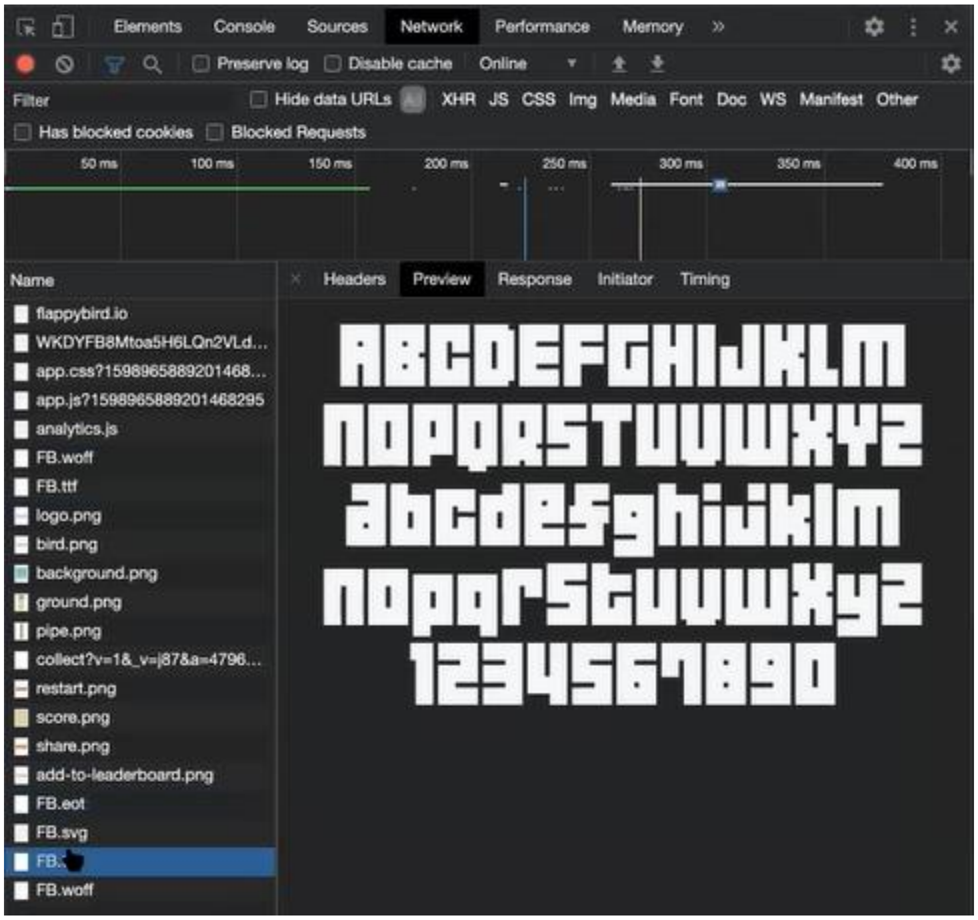
In the developer's page, go to '**Network**' tab.



```
<!DOCTYPE html>
<!--[if lt IE 7]>      <html class="no-js lt-ie9 lt-ie8
lt-ie7"> <![endif]-->
<!--[if IE 7]>         <html class="no-js lt-ie9 lt-
ie8"> <![endif]-->
<!--[if IE 8]>         <html class="no-js lt-ie9"> <!
[endif]-->
<!--[if gt IE 8]><!-->
<html class="no-js">
  <!--<![endif]-->
 ▸<head>…</head>
 ▾<body>
   ▸<nav class="navbar navbar-default">…</nav>
   ▾<div class="container">
      ::before
      <div id="fb-root"></div>
     ▸<div class="mobile-ad">…</div>
     ▾<div class="row">
        ::before
       ▾<div class="col-md-3 col-lg-3 col-lg-offset-0">
         ▾<div class="visible-lg">
            <div id="div-gpt-ad-1396562888498-0" style=
            "width:300px; height:600px; margin-left:
            -11px">
            </div> == $0
          </div>
          <br>
        </div>
       ▸<div class="col-sm-8 col-md-12 col-lg-6">…</div>
       ▸<div class="visible-sm col-sm-4">…</div>
       ▸<div class="col-sm-12 col-md-6 col-md-offset-3
      col-lg-3 col-lg-offset-0">…</div>
        ::after
      </div>
      ::after
    </div>
    <br>
    <br>
  </body>
</html>
```

```
element.style {
   width: 300px;
   height: 600px;
   margin-left: -11px;
}
* {                              app.css?159…201468295:5
   ~~-webkit-box-sizing: border-box;~~
   ~~-moz-box-sizing: border-box;~~
   box-sizing: border-box;
}
div {                           user agent stylesheet
   display: block;
}
Inherited from body
body, html {                     app.css?159…201468295:5
   margin: ▸ 0;
   padding: ▸ 0;
   text-align: center;
   height: 100%;
}
body {                           app.css?159…201468295:5
   font-family: helvetica
      neue,Helvetica,Arial,sans-serif;
   font-size: 14px;
   line-height: 1.42857143;
   color: #c8c8c8;
   background-color: #272b30;
}
Inherited from html.no-js
html {                           app.css?159…201468295:5
   ~~font-size: 62.5%;~~
   -webkit-tap-highlight-color:
      transparent;
}
html {                           app.css?159…201468295:5
   ~~font-family: sans-serif;~~
   ~~-ms-text-size-adjust: 100%;~~
   ~~-webkit-text-size-adjust: 100%;~~
}
Pseudo ::before element
*:before, *:after {              app.css?159…201468295:5
   ~~-webkit-box-sizing: border-box;~~
   ~~-moz-box-sizing: border-box;~~
   box-sizing: border-box;
}
```

In the ‘**Network**’ tab, once on this page, refresh the page and you will be able to see the list of files used for this game.



```
flappybird.io
WKDYFB8Mtoa5H6LQn2VLd...
app.css?1598965889201468...
app.js?1598965889201468295
analytics.js
FB.woff
FB.ttf
logo.png
bird.png
background.png
ground.png
pipe.png
collect?v=1&_v=j87&a=4796...
restart.png
score.png
share.png
add-to-leaderboard.png
FB.eot
FB.svg
FB.
FB.woff
```

Save all these files in your **Flappy-Bird** Folder. Make another folder **Assets** in the Flappy-Bird folder in which we will save all the files that we download from the ‘**Network**’ tab.

We will create a folder **Assets** in the Flappy-Bird folder. In this folder we will create python files; ‘***bird.py***’, ‘***game.py***’, ‘***pipe.py***’ and ‘***scorceboard.py***’

Another folder, ’**Util**’ is created in the Flappy-Bird folder. In this folder, we will create a ‘colors.py’ file and ‘__init__.py’ file. In the ‘main.py’ file we will import Colors from util.colors.

We will create ‘__init__.py’ file in the **Flappy-Bird** Folder as well.

```python
import sys
for util.colors import Colors
import pygame

pygame.init()

WIDTH = 320
HEIGHT = 240

black = 0, 0, 0 # rgb of black is 0, 0, 0
# we can change the color

# to get a screen we use the following command
# we need to have a tuple here
screen = pygame.display.set_mode((WIDTH, HEIGHT))

while True: # this is an infinite loop
    # all the events are in the event
    for event in pygame.event.get():
        # if somebody say quit, we will quit the game
        if event.type == pygame.QUIT: sys.exit()


    screen.fill(colors.black)
    # flip - at the end, it refreshes the game (FPS -
 frames per second)
    # images are very moving very fast
    pygame.display.flip()
```

WIDTH & HEIGHT are constant, so we can create a constant.py file in the util folder. In the constants.py we will give values for width & height.

```python
class Constants:
    width = 500
    height = 500

    game_dimension = (width, height)
```

now the code in the main.py file would be

```python
import sys
from util.constants import Constants
from util.colors import Colors
import pygame


def run():
    pygame.init()

    screen = pygame.display.set_mode(Constants.game_dimension)

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT: sys.exit()

        screen.fill(Colors.black)
        pygame.display.flip()

if __name__ == '__main__':
    pass
```

Do not go deep into anything, just make it work, for that is important.

```python
from entities.background import Background
import sys
from util.constants import Constants
from util.colors import Colors
import pygame
import pathlib

import os


def run():
    pygame.init()
    screen = pygame.display.set_mode(Constants.game_dimention)
    background = Background()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT: sys.exit()

        background.display(screen)
        pygame.display.flip()


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

For the images part,

```
import sys
from util.constants import Constants
from util.colors import Colors
import pygame
import pathlib

import os

def loadbackground():
    # code for background
    current_path = pathlib.Path(__file__).parent.absolute()
    background_path = os.path.joing(current_path, "assets", "im
ages" "background.png")
    background = pygame.image.load(background_path)

def run():

    pygame.init()
    screen = pygame.display.set_mode(Constants.game_dimention)
    load_background()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT: sys.exit()
        screen.blit(background, background.get_rect())
        pygame.display.flip()


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

Commit the changes done.

We will not create a game class

```
class Game:
    def __init__(self):
        self. bird = None
        self.pipes = list()
        self.background = None
        self.score = None

```

Game will also have display that will display all these things.

```
    def display():
        pass
```

we will now create the class Background. Background is the image path,

```
class Background:
    def __init__(self):
        self.image_path = ""

    def display(self):
        pass

```

From the main.py file, we will copy the code; and will make the changes in background.py file.

```
import os
import sys
import pygame

class Background:
    def __init__(self):
        current_path = pathlib.Path(__file__).parent.parent.abs
olute()
        path = os.path.join(current_path, "assets", "images",
"background.png")
        print("background path is", path)
        self.image = pygame.image.load((path))

    def display(self):
        screen.blit(self.image, self.image.get_rect())
```

Now, in the file bird.py file,

```
import os
import sys
import pygame

class Bird:
    def __init__(self):
        current_path = pathlib.Path(__file__).parent.parent.abs
olute()
        path = os.path.join(current_path, "assets", "images", "
bird.png")
        self.image = pygame.image.load(path)

    def display(self, screen):
        screen.blit(self.image, self.image.get_rect())
```

in the main.py file, we will add the bird.display(screen).

```
import sys
from util.constants import Constants
from util.colors import Colors
import pygame
import pathlib

import os

def run():
    pygame.init()
    screen = pygame.display.set_mode(Constants.game_dimention)

    background = Background()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT: sys.exit()

        background.display(screen)
        bird.display(screen)
        pygame.display.flip()

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

The position of bird should be at the center, so in the bird.py file,

```
import os
import sys
import pygame

class Background:
    def __init__(self):
        current_path = pathlib.Path(__file__).parent.parent.abs
olute()
        path = os.path.join(current_path, "assets", "images", "
bird.png")
```

```
        self.image = pygame.image.load(path)

    def display(self, screen):
        bird_rect = self.image.get_rect()
        bird_rect.center = Constants.width//2, Constants.height
//2
        screen.blit(self.image, bird_rect)
```

for the animation part of the bird, we need to move the rectangle image. So, we will give count and the x-coordinate would keep changing (moving)

```
import os
import sys
import pygame

class Background:
    def __init__(self):
        current_path = pathlib.Path(__file__).parent.parent.abs
olute()
        path = os.path.join(current_path, "assets", "images", "
bird.png")
        self.image = pygame.image.load(path)
        self.cnt = 0

    def display(self, screen):
        bird_rect = self.image.get_rect()
        bird_rect.center = Constants.width//2, Constants.height
//2
        self.cnt += 1
        screen.blit(self.image, bird_rect)
```

https://github.com/joshi95/flappy-bird