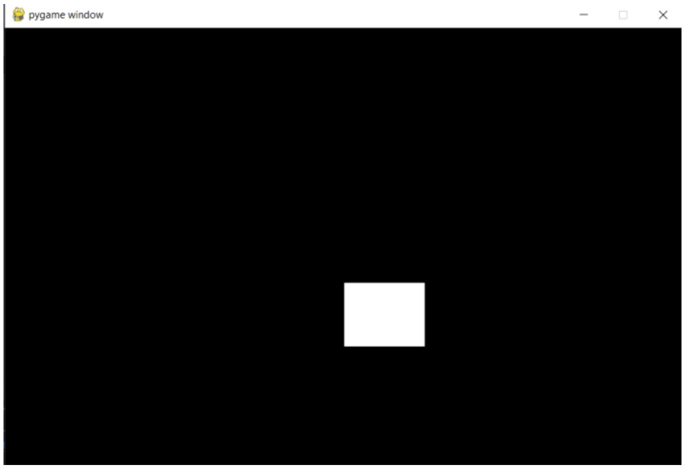
<table>
<tr><td colspan="2">Laboratorium <b>Programowanie w języku Python 2</b>
Wydział Elektrotechniki Automatyki I Informatyki
Politechnika Świętokrzyska</td></tr>
<tr><td>Studia: <b>Stacionarne I stopnia</b></td><td>Kierunek: <b>Informatyka</b></td></tr>
<tr><td>Data wykonania: <b>03.06.2021</b></td><td>Grupa: <b>3ID16B</b></td></tr>
<tr><td>Imię I nazwisko:
<b>Arkadiusz Więcław</b></td><td>Temat ćwiczenia:
<b>Gry, multimedia</b></td></tr>
</table>

Zad 1:

```
# Import the pygame module
import pygame

# Import pygame.locals for easier access to key coordinates
# Updated to conform to flake8 and black standards
from pygame.locals import (

    K_UP,

    K_DOWN,

    K_LEFT,

    K_RIGHT,

    K_ESCAPE,

    KEYDOWN,

    QUIT,

)

# Define constants for the screen width and height
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600

# Define a player object by extending pygame.sprite.Sprite
# The surface drawn on the screen is now an attribute of 'player'
class Player(pygame.sprite.Sprite):

    def __init__(self):

        super(Player, self).__init__()

        self.surf = pygame.Surface((95, 75))

        self.surf.fill((255, 255, 255))
        self.rect = self.surf.get_rect()

    # Move the sprite based on user keypresses
    def update(self, pressed_keys):

        if pressed_keys[K_UP]:
            self.rect.move_ip(0, -5)

        if pressed_keys[K_DOWN]:
            self.rect.move_ip(0, 5)

        if pressed_keys[K_LEFT]:
            self.rect.move_ip(-5, 0)

        if pressed_keys[K_RIGHT]:
            self.rect.move_ip(5, 0)

# Initialize pygame
pygame.init()


# Create the screen object
# The size is determined by the constant SCREEN_WIDTH and
SCREEN_HEIGHT
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))

# Instantiate player. Right now, this is just a rectangle.
player = Player()

# Variable to keep the main loop running
running = True

# Main loop
while running:

    # for loop through the event queue
    for event in pygame.event.get():

        # Check for KEYDOWN event
        if event.type == KEYDOWN:

            # If the Esc key is pressed, then exit the main loop
            if event.key == K_ESCAPE:

                running = False

        # Check for QUIT event. If QUIT, then set running to false.
        elif event.type == QUIT:

            running = False

    # Get all the keys currently pressed
    pressed_keys = pygame.key.get_pressed()

    # Update the player sprite based on user keypresses
    player.update(pressed_keys)

    # Fill the screen with black
    screen.fill((0, 0, 0))

    # Draw the player on the screen
    screen.blit(player.surf, (SCREEN_WIDTH/2, SCREEN_HEIGHT/2))

    # Update the display
    pygame.display.flip()
```

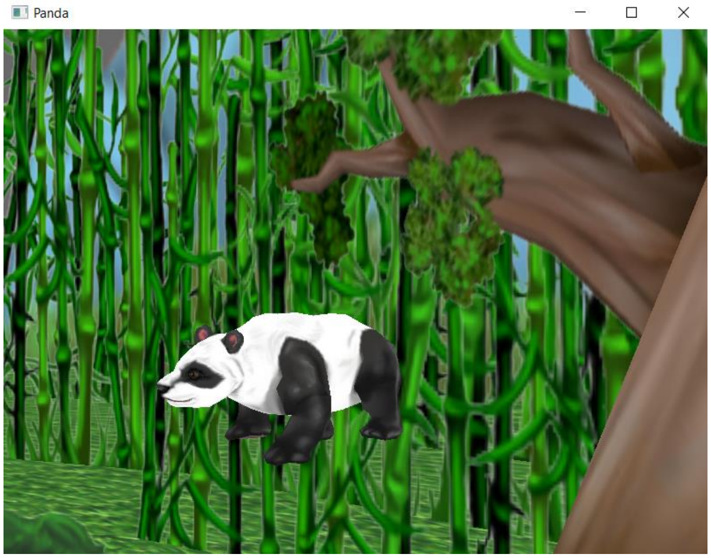
Wynik:



Zad 2:

```
from math import pi, sin, cos
from direct.showbase.ShowBase import ShowBase
from direct.task import Task
from direct.actor.Actor import Actor
from direct.interval.IntervalGlobal import Sequence
from panda3d.core import Point3

"""
Przyklad demostrujace dzialanie panda3d za przykładzie chodzącej
pandy.
"""
class MyApp(ShowBase):
    def __init__(self):
        ShowBase.__init__(self)

        # Disable the camera trackball controls.
        self.disableMouse()

        # Load the environment model.
        self.scene = self.loader.loadModel("models/environment")
        # Reparent the model to render.
        self.scene.reparentTo(self.render)
        # Apply scale and position transforms on the model.
        self.scene.setScale(0.75, 0.35, 0.85)
        self.scene.setPos(-8, 43, -12)

        # Add the spinCameraTask procedure to the task manager.
        self.taskMgr.add(self.spinCameraTask, "SpinCameraTask")

        # Load and transform the panda actor.
        self.pandaActor = Actor("models/panda-model",
                                {"walk": "models/panda-walk4"})
        self.pandaActor.setScale(0.005, 0.005, 0.005)
        self.pandaActor.reparentTo(self.render)
        # Loop its animation.
        self.pandaActor.loop("walk")

        # Create the four lerp intervals needed for the panda to
        # walk back and forth.
        posInterval1 = self.pandaActor.posInterval(13,
                                                   Point3(0, -10, 0),
                                                   startPos=Point3(0,
10, 0))
        posInterval2 = self.pandaActor.posInterval(13,
                                                   Point3(0, 10, 0),
                                                   startPos=Point3(0,
-10, 0))
        hprInterval1 = self.pandaActor.hprInterval(3,
                                                   Point3(180, 0, 0),
                                                   startHpr=Point3(0,
0, 0))
        hprInterval2 = self.pandaActor.hprInterval(3,
                                                   Point3(0, 0, 0),

startHpr=Point3(180, 0, 0))

        # Create and play the sequence that coordinates the intervals.
        self.pandaPace = Sequence(posInterval1, hprInterval1,
                                  posInterval2, hprInterval2,
                                  name="pandaPace")
        self.pandaPace.loop()

    # Define a procedure to move the camera.
    def spinCameraTask(self, task):
        angleDegrees = task.time * 6.0
        angleRadians = angleDegrees * (pi / 180.0)
        self.camera.setPos(20 * sin(angleRadians), -20 *
cos(angleRadians), 3)
        self.camera.setHpr(angleDegrees, 0, 0)
        return Task.cont


app = MyApp()
app.run()
```

Wynik:



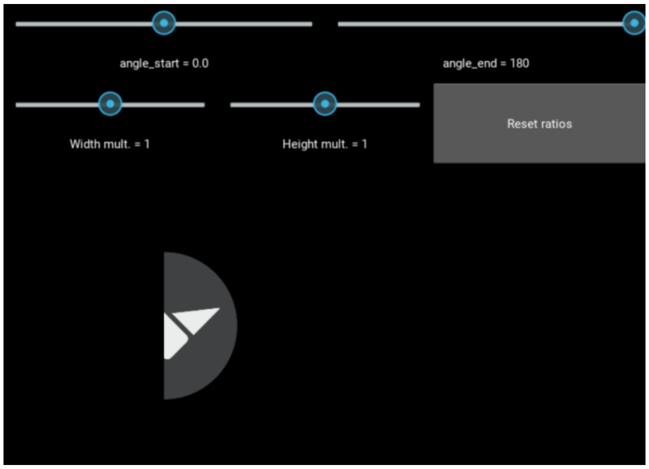
Zad 3:

```
'''
Ten przykład ćwiczy rysowanie okręgu (elipsy).Suwaki na
na górze ekranu z logo Kivy pod nim. Suwaki kontrolują
początek i koniec kąta oraz skale wysokości i szerokości. Jest
przycisk
aby zresetować suwaki. Logo użyte do obrazu tła koła to
z katalogu kivy/data. Cały przykład jest zakodowany w
kv opis języka.
'''

from kivy.app import App
from kivy.lang import Builder

kv = '''
BoxLayout:
    orientation: 'vertical'
    BoxLayout:
        size_hint_y: None
        height: sp(100)
        BoxLayout:
            orientation: 'vertical'
            Slider:
                id: e1
                min: -360.
                max: 360.
            Label:
```

```
            text: 'angle_start = {}'.format(e1.value)
    BoxLayout:
        orientation: 'vertical'
        Slider:
            id: e2
            min: -180.
            max: 180.
            value: 180
        Label:
            text: 'angle_end = {}'.format(e2.value)

BoxLayout:
    size_hint_y: None
    height: sp(100)
    BoxLayout:
        orientation: 'vertical'
        Slider:
            id: wm
            min: 0
            max: 2
            value: 1
        Label:
            text: 'Width mult. = {}'.format(wm.value)
    BoxLayout:
        orientation: 'vertical'
        Slider:
            id: hm
            min: 0
            max: 2
            value: 1
        Label:
            text: 'Height mult. = {}'.format(hm.value)
    Button:
        text: 'Reset ratios'
        on_press: wm.value = 1; hm.value = 1

FloatLayout:
    canvas:
        Color:
            rgb: 1, 1, 1
        Ellipse:
            pos: 100, 100
```

```
                size: 200 * wm.value, 201 * hm.value
                source: 'data/logo/kivy-icon-512.png'
                angle_start: e1.value
                angle_end: e2.value

'''


class CircleApp(App):
    def build(self):
        return Builder.load_string(kv)


CircleApp().run()
```

Wynik:



angle_start = 0.0

angle_end = 180

Width mult. = 1

Height mult. = 1