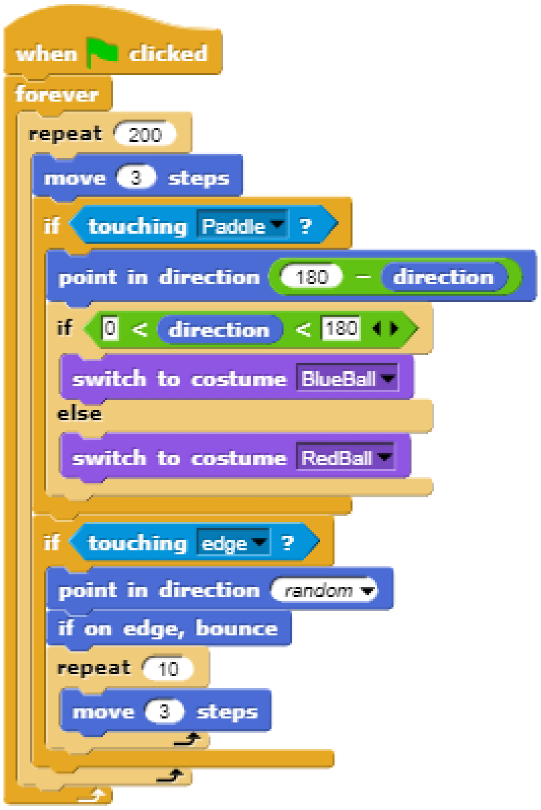
# From Snap! to Python: Ball and Paddle game

UBMS Snap! Programming Summer 2023

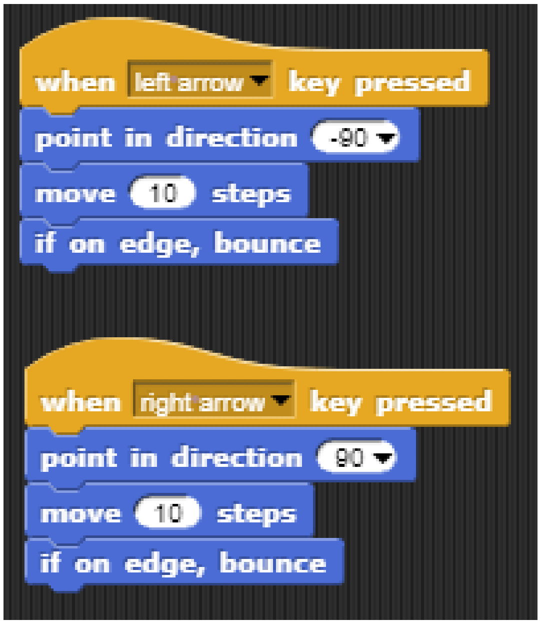
July 10, 2023

## Ball and paddle game in Python

- Games and animations are no small feat in Python. Compare the code for the modified ball-and-paddle game in Snap! with the Python Turtle version (in replit.com):



```
when ⚑ clicked
forever
  repeat (200)
    move (3) steps
    if <touching [Paddle ▾] ?>
      point in direction ((180) − (direction))
      if <[0] < (direction) < [180] ◂▸>
        switch to costume [BlueBall ▾]
      else
        switch to costume [RedBall ▾]
    if <touching [edge ▾] ?>
      point in direction (random ▾)
      if on edge, bounce
      repeat (10)
        move (3) steps
```



```
when left arrow key pressed
point in direction -90
move 10 steps
if on edge, bounce

when right arrow key pressed
point in direction 90
move 10 steps
if on edge, bounce
```

- Here's also a version with the `PyGame` library in replit.com (but I haven't got it work properly yet).

## Turtle graphics

- Turtle graphics is a re-implementation of 1967 graphics for kids in Python that works much like the Snap! turtle.
- At the start, the turtle is positioned at (0,0) of an x-y-grid. The turtle draws as it moves across the canvas:

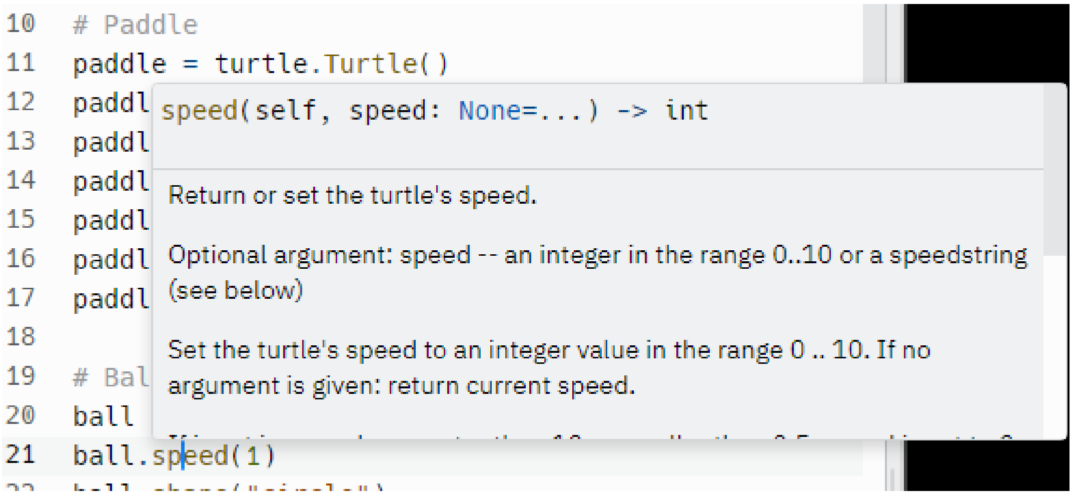
```
import turtle as t
s = t.getscreen() # build the window
t.right(90)
t.forward(100)
t.left(90)
t.backward(100)
t.goto(100,100)
t.delay(100)
t.circle(60)
t.delay(100)
t.dot(20)
```

- This game will not run in Colab because Turtle relies on tkinter, a Python interface to the Tcl/Tk GUI toolkit. This is a standard on Linux but as the error message in Colab shows, Colab cannot direct its output to your screen to open a separate window:

```
TclError: no display name and no $DISPLAY environment variable
```

## Codealong in the REPL

- Open a new REPL in replit.com and name it `BallAndPaddle`.
- Enter all code below in the `main.py` panel.
- Run code intermittently (Python is interpreted!) and click on any command for a docstring (short help information): in the image for the `ball.speed(1)` function call.



```
# Paddle
paddle = turtle.Turtle()
paddl
paddl
paddl
paddl
paddl
paddl

# Bal
ball
ball.speed(1)
```

```
speed(self, speed: None=...) -> int
```

Return or set the turtle's speed.

Optional argument: speed -- an integer in the range 0..10 or a speedstring (see below)

Set the turtle's speed to an integer value in the range 0 .. 10. If no argument is given: return current speed.

## Import libraries

- We need the `turtle` and the `random` library for the ball moves.

```
import turtle
import random
```

## Create a window

- To create a window, we define a variable `win` and give it some properties: `title`, background color, and windows size (800x600):

```
win = turtle.Screen()
win.title("Turtle Paddle Ball Game")
win.bgcolor("black")
win.setup(width=800,height=600)
```

## Create a paddle and initialize it

- The paddle is a white turtle (the basic sprite) of rectangular shape that sits still with `speed(0)`.
- Also, we do not want the paddle to draw while it's moving (`penup`) and go to (0,-250) at the start:

```
paddle = turtle.Turtle()
paddle.speed(0) # paddle sits still
paddle.shape("square")
paddle.color("white")
paddle.shapesize(stretch_wid=1,stretch_len=5)  # rectangle
paddle.penup()   # do not draw when moving
paddle.goto(0,-250)  # paddle's initial location
```

## Create a ball and initialize it

- The ball also starts as a turtle, shaped like a white circle, that moves slow with `speed(1)`.
- Also, we don't want the ball to draw while it's moving (`penup`) and we want it to go to a random (x,y) position at the start.
- Lastly, we define ball velocity as a function call: whenever `ball.dx` or `ball.dy` are called, `random.choice` picks a velocity in (-2,2).

```
ball = turtle.Turtle()
ball.speed(1) # speed is slowest
ball.shape("circle")
ball.color("white")
ball.penup()   # do not draw when moving
ball.goto(0,0)   # ball’s initial location
ball.dx = random.choice((-2, 2))  # Ball’s x velocity
ball.dy = random.choice((-2, 2))  # Ball’s y velocity
```

## Moving the paddle (if on edge, bounce)

- Equivalent to making a block in Snap!, we define functions in Python to move the paddle to the left or right.

- the `if` condition checks if the x-coordinate of the paddle (`paddle.xcor()`) is greater than -350 or smaller than 350. If it is, the paddle is moved to the left or right to prevent it from wandering off stage.

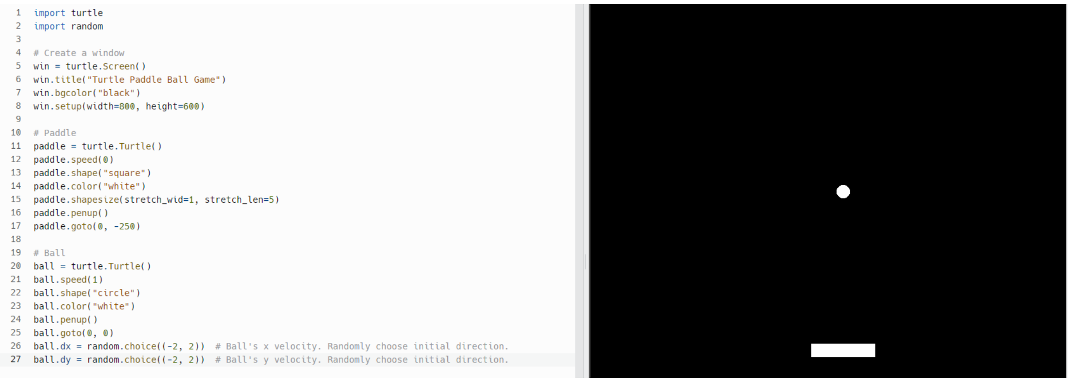
- The functions are then bound to the left and the right arrow keys:

```
def paddle_left(): # prevent paddle from moving off to the left
    x = paddle.xcor()
    if x > -350:
        x -= 20
        paddle.setx(x)

def paddle_right():
    x = paddle.xcor()
    if x < 350:
        x += 20
        paddle.setx(x)

win.listen()   # keyboard bindings - window ’listens’
win.onkeypress(paddle_left, "Left")
win.onkeypress(paddle_right, "Left")
```

- Intermediate screen with the code so far: you can see ball and paddle and move the paddle left and right.



```
import turtle
import random

# Create a window
win = turtle.Screen()
win.title("Turtle Paddle Ball Game")
win.bgcolor("black")
win.setup(width=800, height=600)

# Paddle
paddle = turtle.Turtle()
paddle.speed(0)
paddle.shape("square")
paddle.color("white")
paddle.shapesize(stretch_wid=1, stretch_len=5)
paddle.penup()
paddle.goto(0, -250)

# Ball
ball = turtle.Turtle()
ball.speed(1)
ball.shape("circle")
ball.color("white")
ball.penup()
ball.goto(0, 0)
ball.dx = random.choice((-2, 2))  # Ball's x velocity. Randomly choose initial direction.
ball.dy = random.choice((-2, 2))  # Ball's y velocity. Randomly choose initial direction.
```

## Main game loop

- For the main game loop we use an infinite `while True:` loop.
- We update the screen at the start of each iteration: `win.update`.
- We set a new ball position based on a random velocity.
- We reset ball position depending on the position on the stage - in particular, reverse the ball direction at the top, and when it hits bottom.
- We define ball + paddle collisions and paint the ball blue if is is moving to the right (`ball.dx >0`) and red if it is moving to the left (`ball.dx <0`) after the collision.
- The code:

```
while True:
    win.update()

    # Ball movement
    ball.setx(ball.xcor() + ball.dx)
    ball.sety(ball.ycor() + ball.dy)

    # Border checking for ball
    if ball.ycor() > 290:
        ball.sety(290)
        ball.dy *= -1  # Reverse the ball direction
    if ball.ycor() < -290:
        ball.goto(0, 0)  # Reset ball position if it hits the bottom
```

```
            ball.dy *= -1
        if ball.xcor() > 390:
            ball.setx(390)
            ball.dx *= -1
        if ball.xcor() < -390:
            ball.setx(-390)
            ball.dx *= -1

         # Paddle and ball collisions
         if (ball.dx > 0) and (350 > paddle.xcor() - 50 < ball.xcor() < paddle.xcor()
             ball.color("blue")
             ball.dy *= -1
         elif (ball.dx < 0) and (350 > paddle.xcor() - 50 < ball.xcor() < paddle.xcor
             ball.color("red")
             ball.dy *= -1
```

## Full program

See also: replit.com

The code has 55 command + 20 comment lines:

```
import turtle
import random

# Create a window
win = turtle.Screen()
win.title("Turtle Paddle Ball Game")
win.bgcolor("black")
win.setup(width=800, height=600)

# Paddle
paddle = turtle.Turtle()
paddle.speed(0)
paddle.shape("square")
paddle.color("white")
paddle.shapesize(stretch_wid=1, stretch_len=5)
paddle.penup()
paddle.goto(0, -250)

# Ball
```

```
ball = turtle.Turtle()
ball.speed(1)
ball.shape("circle")
ball.color("white")
ball.penup()
ball.goto(0, 0)
ball.dx = random.choice((-2, 2))  # Ball’s x velocity. Randomly choose initial directi
ball.dy = random.choice((-2, 2))  # Ball’s y velocity. Randomly choose initial directi

# Function to move the paddle
def paddle_left():
    x = paddle.xcor()
    if x > -350:
        x -= 20
        paddle.setx(x)

def paddle_right():
    x = paddle.xcor()
    if x < 350:
        x += 20
        paddle.setx(x)

# Keyboard bindings
win.listen()
win.onkeypress(paddle_left, "Left")
win.onkeypress(paddle_right, "Right")

# Main game loop
while True:
    win.update()

    # Ball movement
    ball.setx(ball.xcor() + ball.dx)
    ball.sety(ball.ycor() + ball.dy)

    # Border checking for ball
    if ball.ycor() > 290:
        ball.sety(290)
        ball.dy *= -1  # Reverse the ball direction
    if ball.ycor() < -290:
```

```
    ball.goto(0, 0)  # Reset ball position if it hits the bottom
    ball.dy *= -1
if ball.xcor() > 390:
    ball.setx(390)
    ball.dx *= -1
if ball.xcor() < -390:
    ball.setx(-390)
    ball.dx *= -1

# Paddle and ball collisions
if (ball.dx > 0) and (350 > paddle.xcor() - 50 < ball.xcor() < paddle.xcor() + 50)
    ball.color("blue")
    ball.dy *= -1
elif (ball.dx < 0) and (350 > paddle.xcor() - 50 < ball.xcor() < paddle.xcor() + 50
    ball.color("red")
    ball.dy *= -1
```