ACTIVITY AS YOU ENTER [L1]

What will be printed if you run the following code snippet?

```
max = 5
name = "Sir Cinnabun Thumpington the Third"
count = 0

for letter in range(len(name)):
  if letter in "aeiou":
    if count < max:
      print(letter)
      count += 1
```

ANIMATION & INTERACTIVITY (LECTURE)

LOGISTICS

- HW01 due on Monday
- This is your first homework with a *presentation*
  - Presentations will be on Weds & Thurs
  - You will be expected to talk about your project code, and complete and explain a question from your last checkpoint
  - You can sign up for a presentation slot here (will also be linked on Ed)
  - If you MISS or are 5+ minutes LATE to your presentation, you may reschedule if there are spots, your score will be capped at 80%
    - if there are no available spots or you can't make it --> 0
  - We will drop your lowest presentation grade
- Office hours are your friend! Hooray for office hours!

LEARNING OBJECTIVES

- Write PennDraw programs that create moving images
- Get introduced to a basic form of iteration with the while loop
- Write programs that react in real time to user inputs
  - Understand how to check for & use mouse input with PennDraw
  - Understand how to check for & use keyboard input with PennDraw

ANIMATION IN PENNDRAW

LAST TIME: ANIMATION AS A FLIPBOOK MODEL

- Draw many images per second
- Each image called a **frame**
- In penndraw, each frame is a single image drawn inside a loop

DRAWING FRAMES IN PENNDRAW

```
import penndraw as pd
# SETUP: This code is run just one time!
pd.set_canvas_size(500, 500)
x_center = 0.5
pd.set_pen_color(pd.HSS_BLUE)
# ANIMATION LOOP: This code is run many times per second,
# over and over and over and over again.
while x_center < 1:
    pd.clear()
    pd.filled_square(x_center, 0.5, 0.1)
    x_center += 0.01
    if x_center > 1 + half_side:
        x_center = -half_side
    pd.advance() # Necessary at the end of the loop
```

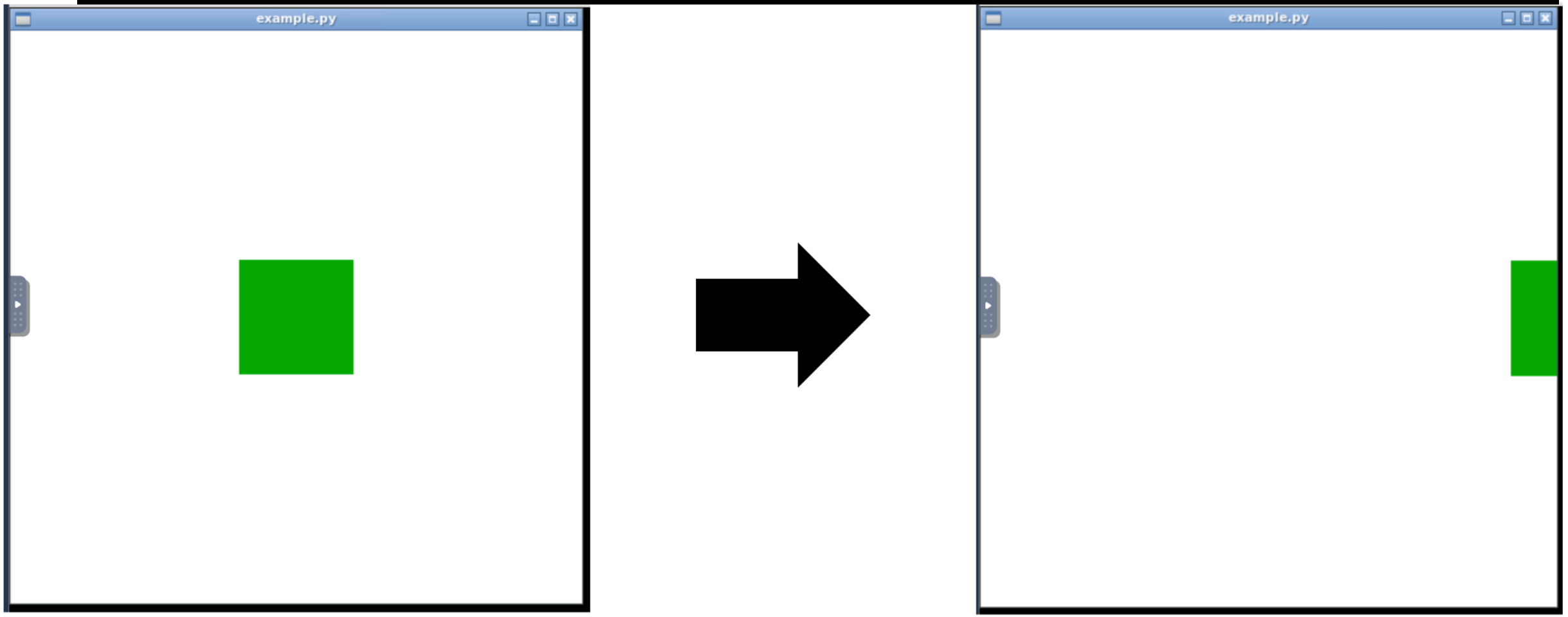
EXAMPLE ANIMATION LOOP: SLIDING SQUARE

Produces a square that slides left-to-right across the canvas.

```
import penndraw as pd
x_center = 0.5 # SETUP
while x_center < 2:
    pd.clear()                                  # 1. clear the screen
    pd.filled_square(x_center, 0.5, 0.1)  # 2. draw this frame
    x_center += 0.1                            # 3. update shapes for next fra
    pd.advance()                                # 4. pd.advance()
```

WHAT HAPPENS WHEN X_CENTER > 1?

WHAT HAPPENS WHEN X_CENTER > 1?



CONTROLLING THE SLIDE

```
import penndraw as pd
x_center = 0.5 # SETUP
while True:
    pd.clear()                                  # 1. clear the screen
    pd.filled_square(x_center, 0.5, 0.1)        # 2. draw this frame
    x_center += 0.01                            # 3. update shapes for next fr
    pd.advance()                                # 4. pd.advance()
```

After a while, `x_center` will be very big.

- If `x_center` is much bigger than `1.0`, the square won't be visible at all!
- Want to add some logic to make sure the square resets after a while

ACTIVITY [L2]

If we're always drawing a particular square using this line:

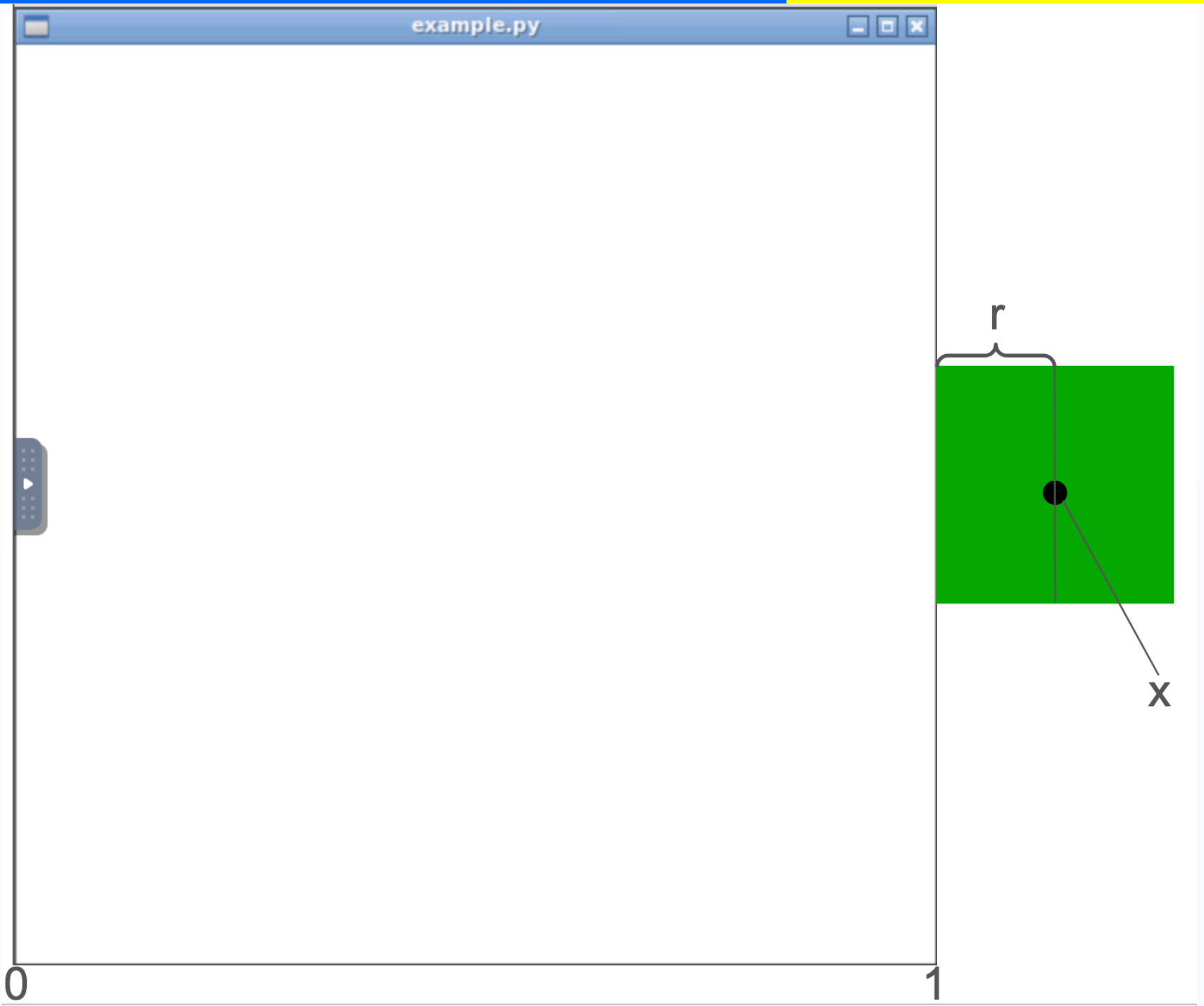
```
pd.square(x, 0.5, r)
```

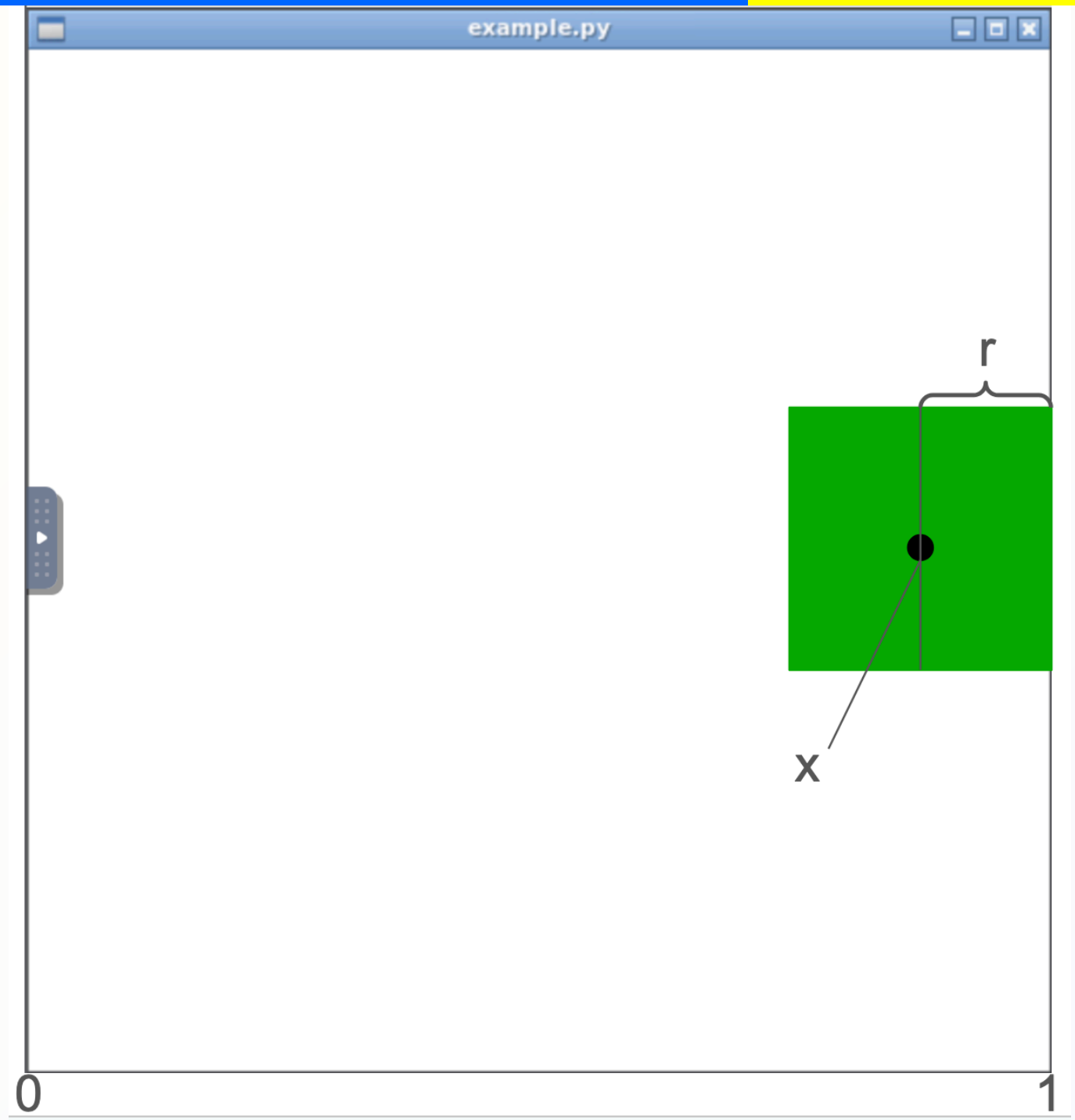
In terms of `x` and `r`,

...write a boolean expression that will be `True` when the square is completely off of the screen to the right.

...write a boolean expression that will be `True` as soon as any part of the square is off the screen to the right.

What is the value of `x` that would draw a square which has its *right* edge just barely touching the *left* boundary of the canvas?


example.py
r
x
0
1


example.py
r
x
0
1

CONTROLLING THE SLIDE

What does "off the screen" mean?

- Happens when a shape is all the way off the top, bottom, left, or right sides of the screen.
- If the square is always heading to the right, it'll fall off the right side
- Since the square has a `half_length` of `0.1`, the coordinate of its *left side* is always `x_center - 0.1`.
- ➡️ the square is offscreen when `x_center - 0.1 > 1.0`

MAKING THE SQUARE RESET TO THE LEFT ONCE IT DISAPPEARS:

- After we update the square, check if it's offscreen
- If the square is offscreen, move it all the way to the left of the screen
- If the square is not offscreen, don't do anything else extra

```
import penndraw as pd
x_center = 0.5 # SETUP
while True:
    pd.clear()                                  # 1. clear the screen
    pd.filled_square(x_center, 0.5, 0.1)  # 2. draw this frame
    x_center += 0.01                            # 3. update shapes for next fr
    if x_center - 0.1 > 1.0:
      x_center = -0.1
    pd.advance()                                # 4. pd.advance()
```

MOUSE INPUT

CLICKING INTO PLACE

PennDraw includes a few tools useful for handling cursor position & clicking:

FUNCTION	RETURN TYPE	DESCRIPTION
pd.mouse_pressed()	bool	Returns *True* if the mouse is being held this frame.
pd.mouse_x()	float	Returns the **x coordinate** of the mouse's current location, e.g. *0.9* or *0.1443*
pd.mouse_y()	float	Returns the **y coordinate** of the mouse's current location, e.g. *0.9* or *0.1443*

CLICK COUNTER (AS SEEN IN VIDEO)

```
import penndraw as pd
num_clicks = 0;
while num_clicks < 100:
  pd.clear()
  pd.text(0.5, 0.5, f"Number of Clicks: {num_clicks}")
  if pd.mouse_pressed():
    num_clicks = num_clicks + 1
  pd.advance()
```

Each frame, if we click the mouse, increment `num_clicks`*, up to 100 clicks.*

ACTIVITY: FOLLOWING SQUARE [C1]

Remember what we've learned about mouse input

`pd.mouse_x()` `pd.mouse_y()` `pd.mouse_pressed()`

Help complete the following code so that the square moves so that its center follows your mouse.

FOLLOWING SQUARE

```
import penndraw as pd

x_center = 0.5
y_center = 0.5
half_side = 0.1
num_clicks = 0

while num_clicks < 100:
  pd.clear()
  x_center = _____________
  y_center = _____________
  if pd.mouse_pressed:
    num_clicks += 1
  pd.filled_square(x_center, y_center, half_side)
```

FOLLOWING SQUARE

```
import penndraw as pd

x_center = 0.5
y_center = 0.5
half_side = 0.1
num_clicks = 0

while num_clicks < 100:
  pd.clear()
  x_center = pd.mouse_x()    # Ask for the x-coordinate of the cursor
  y_center = pd.mouse_y()    # Ask for the y-coordinate of the cursor
  if pd.mouse_pressed():     # Ask whether the mouse is being clicked
    num_clicks += 1
  pd.filled_square(x_center, y_center, half_side)
```

KEYBOARD INPUT

”KEYS” TO SUCCESS

User key presses can also be registered!

FUNCTION	RETURN TYPE	DESCRIPTION
pd.has_next_key_typed()	bool	Returns **True** if a key is currently being held
pd.next_key_typed()	str	Returns the key currently being held down.

🚨 Don't use `next_key_typed()` without checking `has_next_key_typed()` first! 🚨

CHECKING KEYS

The value produced by `pd.next_key_typed()` is a string with a length of one: just a single character.

```
key = pd.next_key_typed()
```

- To see if a specific key was pressed:
  - `if key == "x": ...` or `if key == "!": ...` e.g.
- To see if the key was a lowercase letter:
  - `if "a" <= key <= "z": ...`
- To see if the key was a digit:
  - `if "0" <= key <= "9": ...`

LIGHT SWITCH [L3]

Why does this program crash?

```
import penndraw as pd
on = False
while True:
  if not on:
    pd.clear(pd.BLACK)
  else:
    pd.clear(pd.YELLOW)

  key = pd.next_key_typed()
  if key == "x":
    on = not on
  pd.advance()
```

LIGHT SWITCH [L3]

Why does this program crash? (fixed)

```
import penndraw as pd
on = False
while True:
  if not on:
    pd.clear(pd.BLACK)
  else:
    pd.clear(pd.YELLOW)

  if pd.has_next_key_typed(): <-- we need to check first!
    key = pd.next_key_typed()
    if key == "x":
      on = not on
  pd.advance()
```

RANDOMNESS

IF TIME...

What if our program didn't always draw the same picture each time?

```
import random
print("Picking a random number between 0 and 0.99999...")
my_float = random.random()
print("Picking a random integer between 1 and 100.")
my_int = random.randint(1, 100)
print("my_float:", my_float, "my_int:", my_int)
```

🖨️👇 (for example)

```
Picking a random number between 0 and 0.99999...
Picking a random integer between 1 and 100.
my_float: 0.30258196864839937 my_int: 13
```

PICKING A RANDOM COLOR

How can we fill in the blank with lines of code so that we pick a random color for our square each time?

```
import random
import penndraw as pd

#  PUT SOME CODE HERE!

pd.set_pen_color(red, green, blue)
pd.filled_circle(0.5, 0.5, 0.2)
pd.run()
```

PICKING A RANDOM COLOR

How can we fill in the blank with lines of code so that we pick a random color for our square each time?

```
import random
import penndraw as pd

red = random.randint(0, 255)
green = random.randint(0, 255)
blue = random.randint(0, 255)

pd.set_pen_color(red, green, blue)
pd.filled_circle(0.5, 0.5, 0.2)
pd.run()
```

RANDOM EVENTS

- Each call to `random.random()` gives a result between `0` and `1` where each is equally likely.
- Therefore, there's a 100% chance the number generated is less than `1`
- There's a 90% chance the number generated is less than `0.9`
- There's an 80% chance the number generated is less than `0.8`
- There's an 53.4% chance the number generated is less than `0.534`

➡️ to simulate an event that happens $x\%$ of the time, draw a random number and check if it falls in the range of $(0, \frac{x}{100}]$