Module 5. Lesson 3. OOP. The Simple Paint project

Qualifications

Demonstrate **your knowledge** to begin working on the tasks.

Show that you are ready for brainstorming and project work!

Qualifications

What is an object?

What are properties and methods?

How can we access an object property or method programmatically?

Qualifications

An object

is a set of data and actions that is convenient to perceive as a whole.

An object is said to have properties and be controlled by methods.

Properties	*Methods*
rabbit.speed = 50	rabbit.run()
turtle.speed = 1	turtle.walk()
fish.speed = 30	fish.swim()

Variable placed inside the object.

Function placed inside the object.

How do we add a new property to an existing object?

Qualifications

Creating a new object property

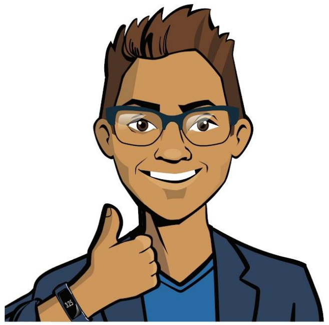
Creating a new property is similar to creating a variable:

Object.property = value

For example, a new property can be set for the Turtle object t:

t.points = 0

Objects have a different scope than functions, so changing their values won't be a problem!



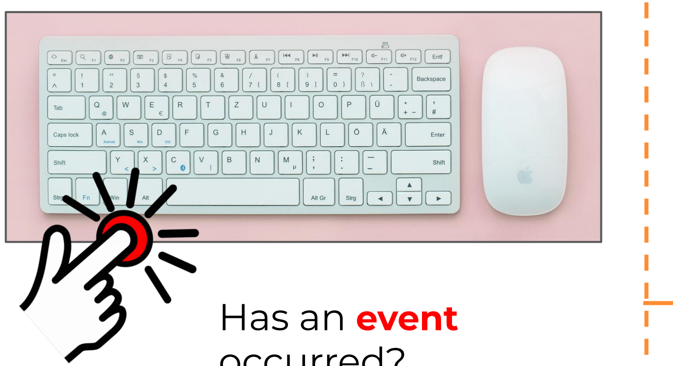
What is considered the “ outside world” of a program?

What is an event?

An event

is information prepared by the execution system about what is happening in the "outside world."

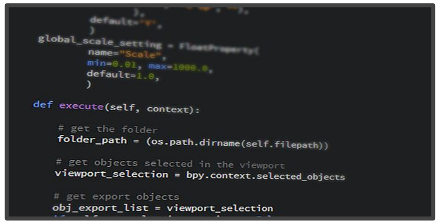
The outside world
is any **equipment** connected to the computer.



Has an **event** occurred?

Execution system:
"An **event** has occurred!"
(*Prepares information about it*).

Running program



```
            default='Y',
            )
    global_scale_setting = FloatProperty(
            name="Scale",
            min=0.01, max=1000.0,
            default=1.0,
            )

    def execute(self, context):

        # get the folder
        folder_path = (os.path.dirname(self.filepath))

        # get objects selected in the viewport
        viewport_selection = bpy.context.selected_objects

        # get export objects
        obj_export_list = viewport_selection
```

Can a program react to an event ?

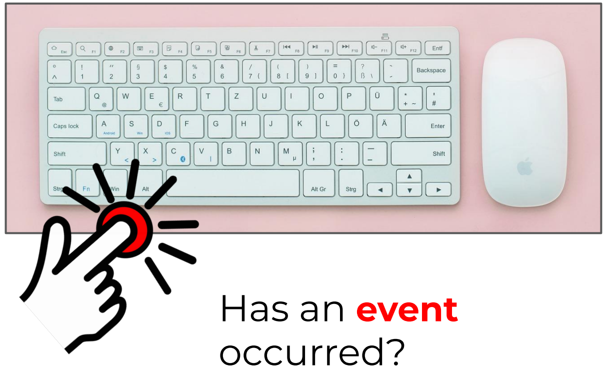
If it can , how do we program that?

To handle an event, the program needs to:

➔ subscribe to an outside world event;

➔ specify in the subscription a handler function the interpreter will call.

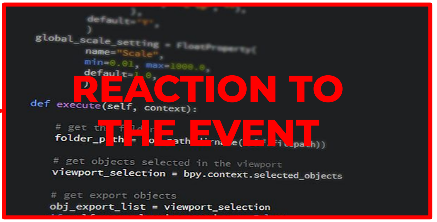
The outside world
is any **equipment** connected to the computer.



Has an **event** occurred?

Execution system:
"An **event** has occurred!"
(*Prepares information about it*).

Running program
"This is an **important event** to me!
I have to **react**."



```
        default='Y',
        )
global_scale_setting = FloatProperty(
        name="Scale",
        min=0.01, max=1000.0,
        default=1.0,
        )

def execute(self, context):

    # get the folder
    folder_path = os.path.dirname(self.filepath))

    # get objects selected in the viewport
    viewport_selection = bpy.context.selected_objects

    # get export objects
    obj_export_list = viewport_selection
```

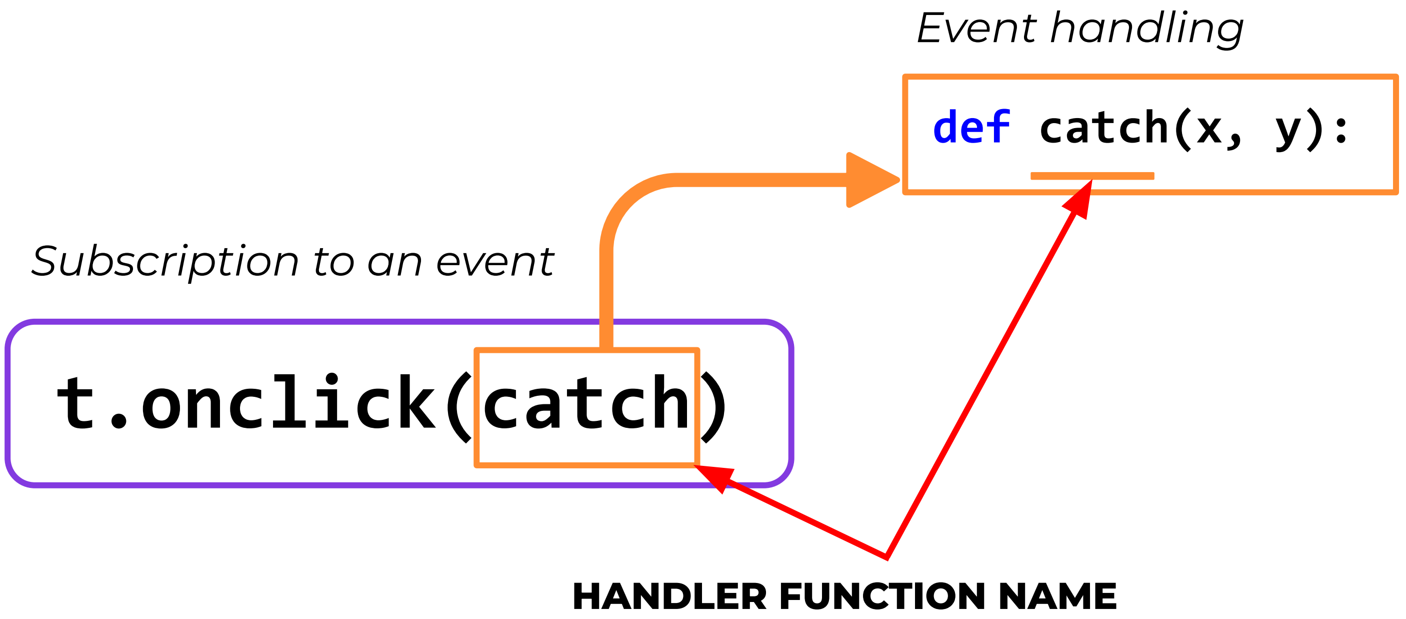
How do we subscribe to the "click on the turtle" event and handle it?

Qualifications

Handling the "click on the turtle" event

To handle a click on an object, we'll create a **catch()** function, whose parameters will be the coordinates of the "caught" turtle.

The location of the click is sent by the execution system **by subscribing to the event**.



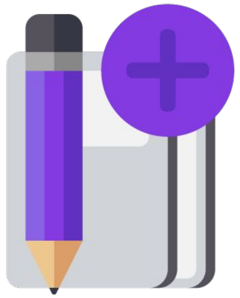
```
def catch(x, y):
```

```
t.onclick(catch)
```

Qualifications confirmed!

Great, you are ready to brainstorm and work on your tasks!

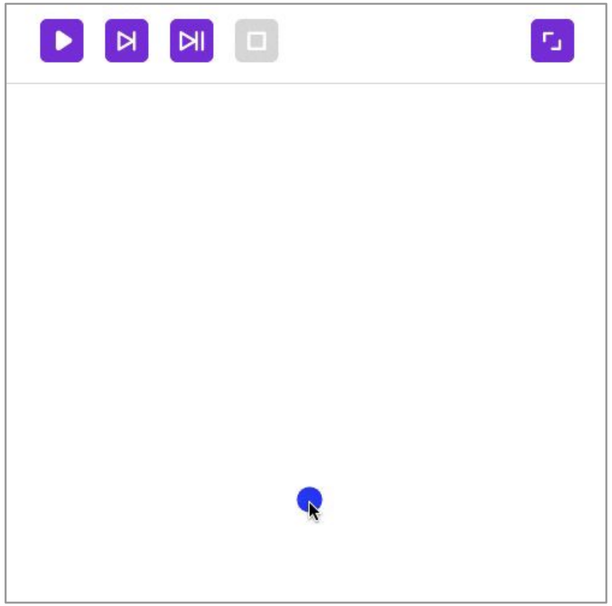
Brainstorming:
Screen objects and working with them



Improving

What event is happening in this picture?

How many times does the user click on the turtle?



Brainstorming

Improving

What event is happening in this picture?

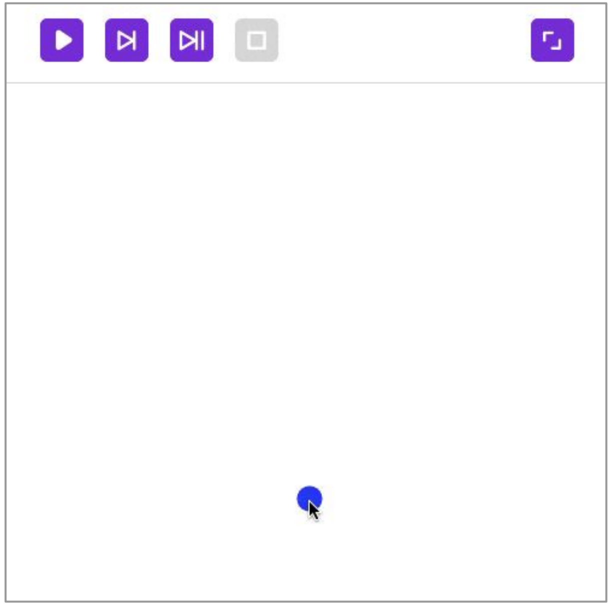
How many times does the user click on the turtle?

What is happening?

- ❑ The user **holds down** the left mouse button on the turtle.
- ❑ The user **moves** the cursor and draws with the turtle.

The program also knows the coordinates of that movement...

Does this seem like a "click on the turtle"?

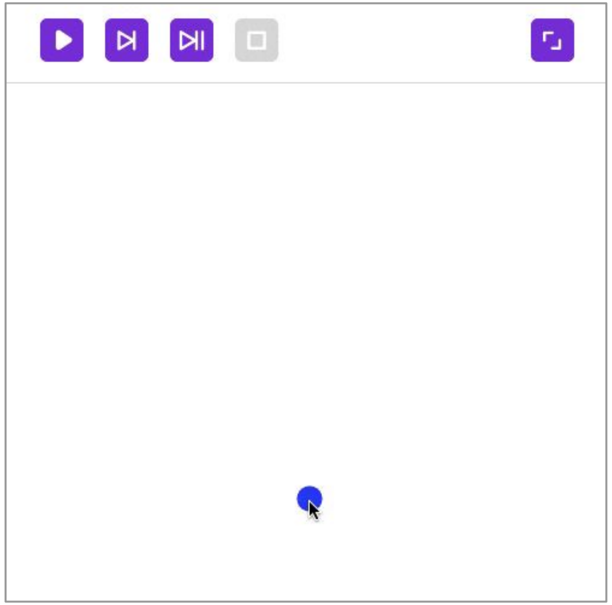


Programming the ability to draw

This is not a "click on the turtle," it’s "**dragging and dropping the turtle**"!

Accordingly:

➔ *a different command is needed* ***to subscribe to the event;***
➔ *a new draw() function needs to be written* ***to handle it****.*



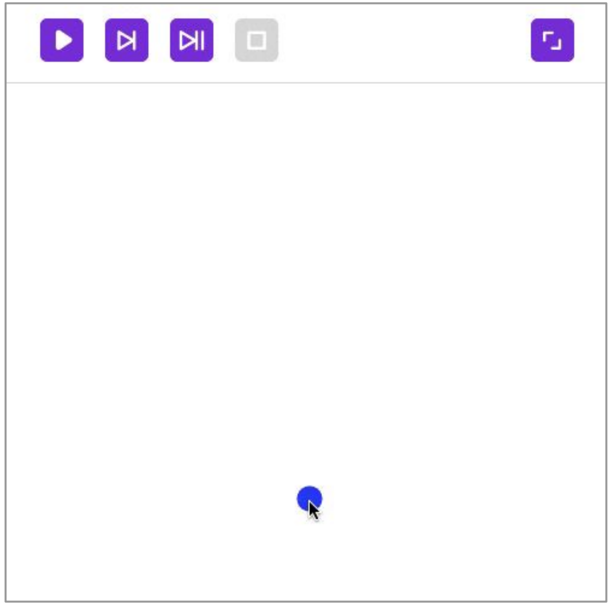
Brainstorming

Programming the ability to draw

This is not a "click on the turtle," it’s "**dragging and dropping the turtle**"!

Accordingly:

➔ *a different command is needed* ***to subscribe to the event;***
➔ *a new draw() function needs to be written* ***to handle it****.*



At every moment, the program knows the coordinates of the turtle's current position — **X and Y**.

Let's use them to draw a line!

Event subscriptions: new command

Command	*Purpose*
`t.onclick(<function_name>)`	Subscribe to **click on the turtle** (requires a function with two parameters)
`t.ondrag(<function_name>)`	Subscribe to **drag and drop the turtle** (requires a function with two parameters)

Event subscription

```
t.ondrag(draw)
```

Event handling

```
def draw(x, y):
```

Brainstorming

Sample program code:

```
from turtle import *


t = Turtle()
t.color('blue')
t.width(5)
t.shape('circle')
t.pendown()
t.speed(3)


def draw(x, y):
  t.goto(x, y)


t.ondrag(draw)
```

Create a Turtle object and set its properties.

Pen down, otherwise there won't be any drawing!

Create a handler function for draw(): when dragging the turtle ***the pen moves to the drop point***.

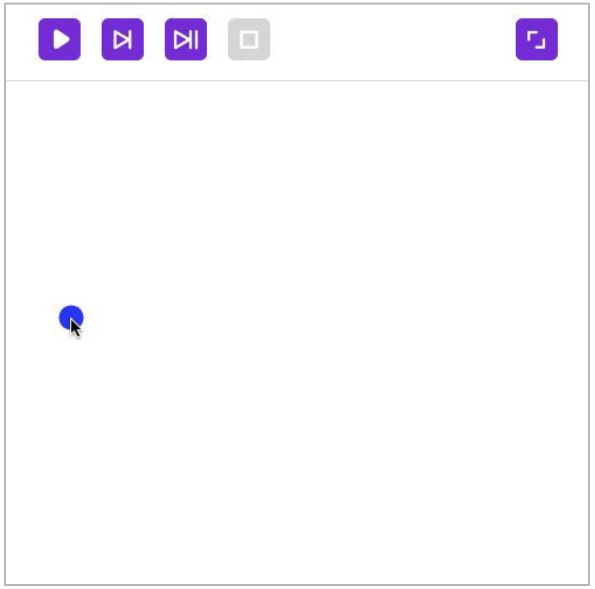
Brainstorming

Programming the ability to draw

Please note that in the expected version of the project, the rendering of <u>multiple</u> objects is permitted!

What event occurs when the pen is moved to a different drawing start point?

How do we handle a <u>click not on the turtle</u>?



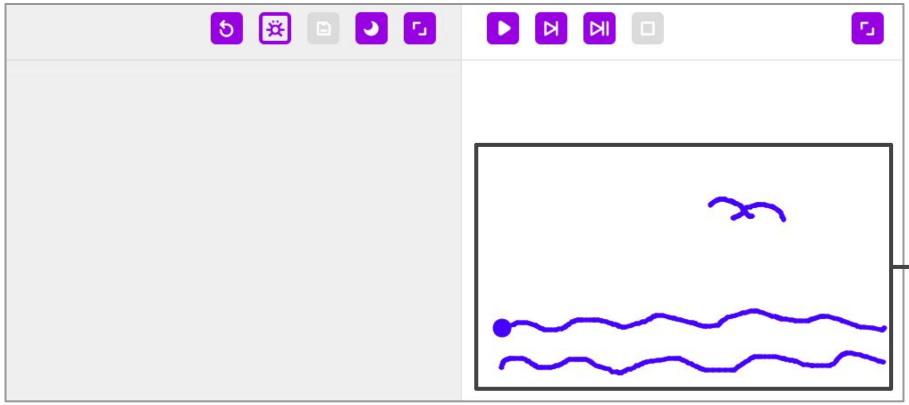
Hint.
Events can involve more than just the turtle.

Screen objects and working with them

The transfer of the drawing point is associated with a "**click on the screen**" event!

A Screen object knows what is happening at any point on the work plane.

Creating a Screen object → `scr = t.getscreen()`



You need the particular screen the turtles are moving on!

Brainstorming

Event subscriptions: new command

Command	*Purpose*
`t.onclick(<function_name>)`	Subscribe to **click on the turtle** (requires a function with two parameters)
`t.ondrag(<function_name>)`	Subscribe to **drag and drop the turtle** (requires a function with two parameters)
`scr.onscreenclick(<function_name>)`	Subscribe to **click on screen** (requires a function with two parameters)

```
def move(x, y):
```

```
scr.onscreenclick(move)
```

Brainstorming

Drawing multiple shapes:

Create a pen for drawing — the Turtle object

The pen dragging handler function draw()

```
def move(x, y):
  t.penup()
  t.goto(x, y)
  t.pendown()

scr = t.getscreen()
scr.onscreenclick(move)

t.ondrag(draw)
```

Create a Turtle object and set its properties.

Pen down, otherwise there won't be any drawing!

Create a Screen object as a ***screen on which the turtle's pen can move.***

A click on the screen (not the turtle!) is handled by the move() function.

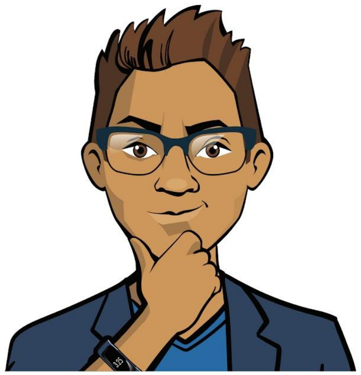
Brainstorming

The task:

Program a **prototype of the Simple Paint graphics editor**.

Implement the drawing of one or more curved lines in one color. Use Turtle and Screen objects.

Use the *documentation* if necessary.



Brainstorming

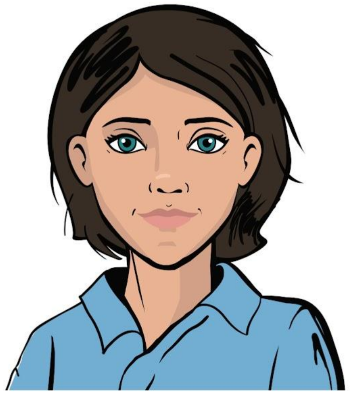
Brainstorming:

Keyboard events

Improving the application

- ➔ add the **ability to select a color** by pressing a key on the keyboard. For example, g for green;
- ➔ program the **drawing of perfectly straight lines** using the arrow keys: Up, Down, Left, Right.

To do this, let's look at subscribing to keyboard events and handling them!



Keyboard events

1. New objects do not need to be introduced to subscribe to keyboard events: **Screen objects** are sufficient.
2. In order for a Screen object to "listen to the keys," the **scr.listen()** command must be introduced. By default, only the mouse is tracked.
3. Subscribing to a keyboard event occurs using the **scr.onkey()** method.
4. We will write our own handler functions.

Brainstorming

Event subscriptions: new command

Command	*Purpose*
`t.onclick(<function_name>)`	Subscribe to **click on the turtle** (requires a function with two parameters)
`t.ondrag(<function_name>)`	Subscribe to **drag and drop the turtle** (requires a function with two parameters)
`scr.listen()`	Command to tell the Screen object to listen to keys
`scr.onkey(<function_name>, <key>)`	Subscribe to **click on key** (the function should have no parameters)

Changing the pen color using keys:

Creating a pen for drawing — the Turtle object

The pen dragging handler function draw()

The moving the pen to another point handler function move()

```
def setGreen():
   t.color('green')


scr = t.getscreen()

scr.listen()
scr.onkey(setGreen,'g')


scr.onscreenclick(move)
t.ondrag(draw)
```

We handle the click on 'g' key with our own setGreen() function.

Before subscribing to the event, ***we indicate that the screen should also track the keys.***

The ability to use other colors is set the same way.

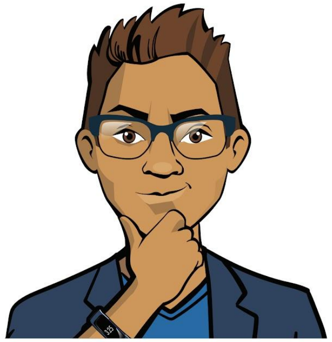
Brainstorming

Drawing using the keyboard

➔ **Subscribing to pressing an arrow key** will be the same as pressing a letter key.

- Arrow up — 'Up'.
- Arrow down — 'Down'.
- Arrow left — 'Left'.
- Arrow right — 'Right'.

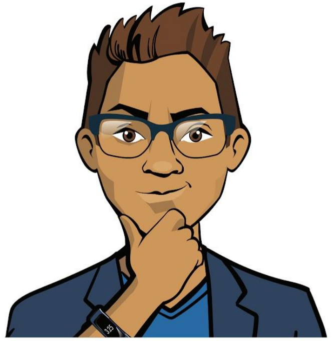
➔ In the handler functions for a single click on an arrow, we describe the **movement of the lowered turtle's pen** in the desired direction **by 5 pixels**.



Drawing using the keyboard

- **Subscribing to pressing an arrow key** will be the same as pressing a letter key.
  - Arrow up — ‘Up’.
  - Arrow down — ‘Down’.
  - Arrow left — ‘Left’.
  - Arrow right — ‘Right’.
- In the handler functions for a single click on an arrow, we describe the **movement of the lowered turtle's pen** in the desired direction **by 5 pixels**.

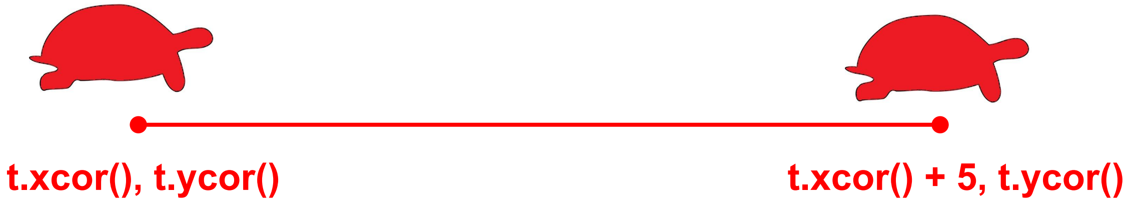
Let’s consider the right arrow key.
How do we move the pen 5 pixels to the right relative to its current position?



Drawing using the keyboard

Let's recall the commands t.xcor() and t.ycor(), which return the current coordinates of an object!

Pressing the Right Arrow key once:



What will the coordinates be when you press the down arrow key?

Which Turtle method will move the turtle to a point with those coordinates?

Brainstorming

Drawing using the keyboard:

Creating a pen for drawing — the Turtle object

The pen dragging handler function draw()

The moving the pen to another point handler function move()

The changing the pen color handler functions

```
def stepRight():
  t.goto(t.xcor() + 5, t.ycor())

scr = t.getscreen()

scr.listen()

scr.onkey(stepRight,'Right')
```

Other event subscriptions

We handle the pressing of the right arrow key with our own stepRight() function.

Note. The shift step can also be defined as a new Turtle property!

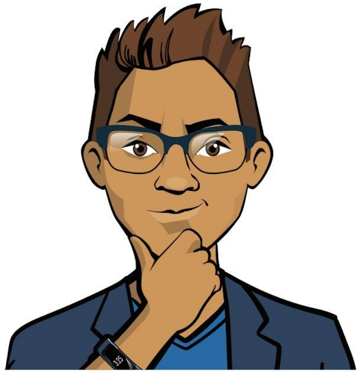
Brainstorming

The task:

Add new functionality to the **prototype of the Simple Paint graphics editor**.

Implement changing the color of the pen by pressing the keyboard keys (g - green, b - blue, etc.).

Program the ability to draw straight lines using the arrow keys.



Brainstorming