Your future studies...

You will have courses that make use of the programming language JavaScript. Mostly in relation to web-development and design.

JavaScript is usually run in the browser. That is unlike the Python programs that you run directly on your operating system.

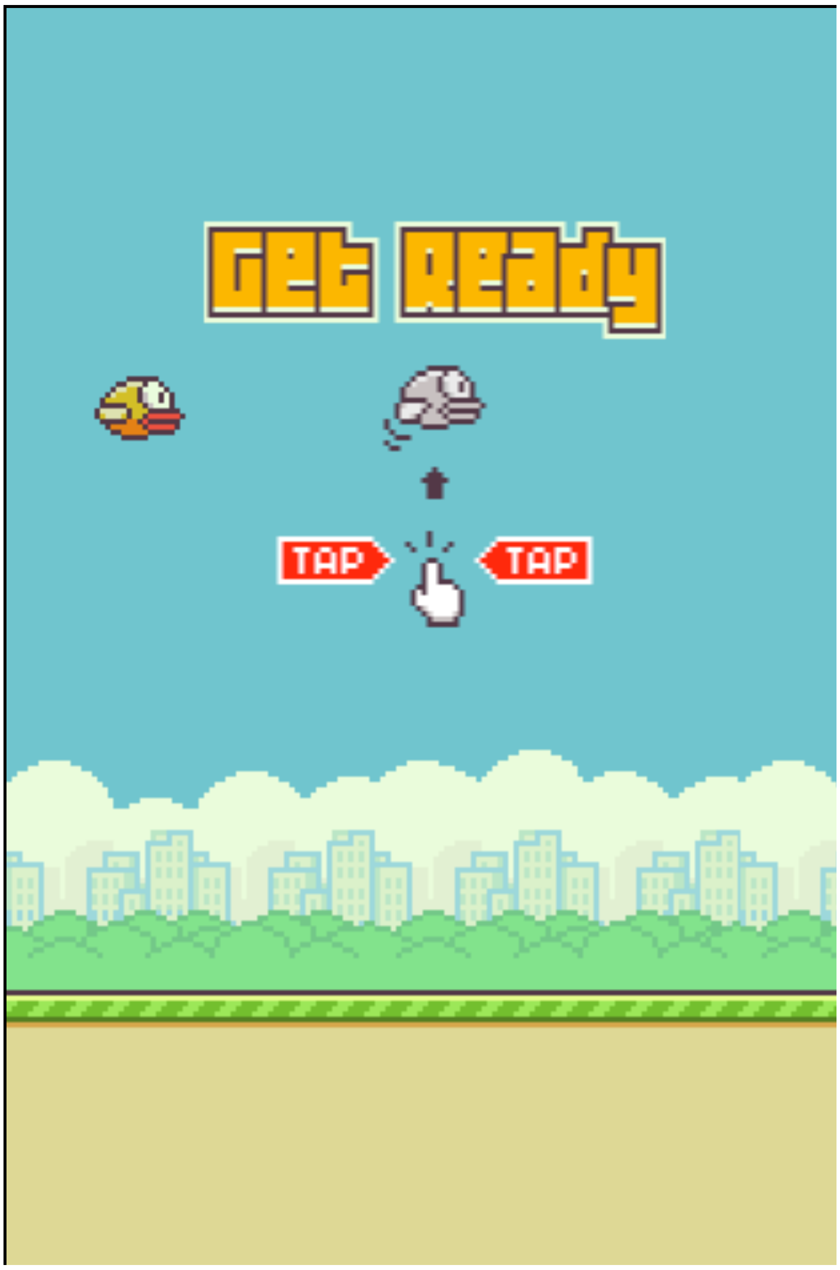
In [ ]:

```

```

In [8]:

```python
from IPython.display import IFrame


url = 'http://itu.dk/people/ropf/flappybird'
IFrame(src=url, width='320', height='480')
```

Out[8]:



JavaScript Syntax

JavaScript has a syntax that is different to Python's.

```
// SELECT CVS
const cvs = document.getElementById("bird");
const ctx = cvs.getContext("2d");

// GAME VARS AND CONSTS
let frames = 0;
const DEGREE = Math.PI/180;

// LOAD SPRITE IMAGE
const sprite = new Image();
sprite.src = "img/sprite.png";

// LOAD SOUNDS
const SCORE_S = new Audio();
SCORE_S.src = "audio/sfx_point.wav";

const FLAP = new Audio();
FLAP.src = "audio/sfx_flap.wav";

const HIT = new Audio();
HIT.src = "audio/sfx_hit.wav";

const SWOOSHING = new Audio();
SWOOSHING.src = "audio/sfx_swooshing.wav";

const DIE = new Audio();
DIE.src = "audio/sfx_die.wav";

// GAME STATE
const state = {
    current : 0,
    getReady : 0,
    game : 1,
    over : 2
}

// START BUTTON COORD
const startBtn = {
    x : 120,
    y : 263,
    w : 83,
    h : 29
```

```
}

// CONTROL THE GAME
cvs.addEventListener("click", function(evt){
    switch(state.current){
        case state.getReady:
            state.current = state.game;
            SWOOSHING.play();
            break;
        case state.game:
            if(bird.y - bird.radius <= 0) return;
            bird.flap();
            FLAP.play();
            break;
        case state.over:
            let rect = cvs.getBoundingClientRect();
            let clickX = evt.clientX - rect.left;
            let clickY = evt.clientY - rect.top;

            // CHECK IF WE CLICK ON THE START BUTTON
            if(clickX >= startBtn.x && clickX <= startBtn.x + startBtn.w && clickY >= startBtn.y && clickY <= startBtn.y + startBtn.h){
                pipes.reset();
                bird.speedReset();
                score.reset();
                state.current = state.getReady;
            }
            break;
    }
});


// BACKGROUND
const bg = {
    sX : 0,
    sY : 0,
    w : 275,
    h : 226,
    x : 0,
    y : cvs.height - 226,

    draw : function(){
        ctx.drawImage(sprite, this.sX, this.sY, this.w, this.h, this.x, this.y, this.w, this.h);

        ctx.drawImage(sprite, this.sX, this.sY, this.w, this.h, this.x + this.w, this.y, this.w, this.h);
    }
```

```
}

// FOREGROUND
const fg = {
    sX: 276,
    sY: 0,
    w: 224,
    h: 112,
    x: 0,
    y: cvs.height - 112,

    dx : 2,

    draw : function(){
        ctx.drawImage(sprite, this.sX, this.sY, this.w, this.h, this.x, this.y, this.w, this.h);

        ctx.drawImage(sprite, this.sX, this.sY, this.w, this.h, this.x + this.w, this.y, this.w, this.h);
    },

    update: function(){
        if(state.current == state.game){
            this.x = (this.x - this.dx)%(this.w/2);
        }
    }
}

// BIRD
const bird = {
    animation : [
        {sX: 276, sY : 112},
        {sX: 276, sY : 139},
        {sX: 276, sY : 164},
        {sX: 276, sY : 139}
    ],
    x : 50,
    y : 150,
    w : 34,
    h : 26,

    radius : 12,

    frame : 0,

    gravity : 0.25,
    jump : 4.6,
    speed : 0,
    rotation : 0,

    draw : function(){
        let bird = this.animation[this.frame];

        ctx.save();
        ctx.translate(this.x, this.y);
        ctx.rotate(this.rotation);
        ctx.drawImage(sprite, bird.sX, bird.sY, this.w, this.h,- this.w/2, - this.h/2, this.w, this.h);

        ctx.restore();
    },

    flap : function(){
        this.speed = - this.jump;
    },

    update: function(){
        // IF THE GAME STATE IS GET READY STATE, THE BIRD MUST FLAP SLOWLY
        this.period = state.current == state.getReady ? 10 : 5;
        // WE INCREMENT THE FRAME BY 1, EACH PERIOD
        this.frame += frames%this.period == 0 ? 1 : 0;
        // FRAME GOES FROM 0 To 4, THEN AGAIN TO 0
        this.frame = this.frame%this.animation.length;

        if(state.current == state.getReady){
            this.y = 150; // RESET POSITION OF THE BIRD AFTER GAME OVER
            this.rotation = 0 * DEGREE;
        }else{
            this.speed += this.gravity;
            this.y += this.speed;

            if(this.y + this.h/2 >= cvs.height - fg.h){
                this.y = cvs.height - fg.h - this.h/2;
                if(state.current == state.game){
                    state.current = state.over;
                    DIE.play();
                }
            }

            // IF THE SPEED IS GREATER THAN THE JUMP MEANS THE BIRD IS FALLING DOWN
            if(this.speed >= this.jump){
                this.rotation = 90 * DEGREE;
                this.frame = 1;
            }else{
                this.rotation = -25 * DEGREE;
            }
        }

    },
    speedReset : function(){
        this.speed = 0;
    }
}

// GET READY MESSAGE
const getReady = {
    sX : 0,
    sY : 228,
    w : 173,
    h : 152,
    x : cvs.width/2 - 173/2,
    y : 80,

    draw: function(){
        if(state.current == state.getReady){
            ctx.drawImage(sprite, this.sX, this.sY, this.w, this.h, this.x, this.y, this.w, this.h);
        }
    }

}

// GAME OVER MESSAGE
const gameOver = {
    sX : 175,
    sY : 228,
    w : 225,
    h : 202,
    x : cvs.width/2 - 225/2,
    y : 90,

    draw: function(){
        if(state.current == state.over){
            ctx.drawImage(sprite, this.sX, this.sY, this.w, this.h, this.x, this.y, this.w, this.h);
        }
    }

}

// PIPES
```

```
const pipes = {
    position : [],

    top : {
        sX : 553,
        sY : 0
    },
    bottom:{
        sX : 502,
        sY : 0
    },

    w : 53,
    h : 400,
    gap : 85,
    maxYPos : -150,
    dx : 2,

    draw : function(){
        for(let i  = 0; i < this.position.length; i++){
            let p = this.position[i];

            let topYPos = p.y;
            let bottomYPos = p.y + this.h + this.gap;

            // top pipe
            ctx.drawImage(sprite, this.top.sX, this.top.sY, this.w, this.
h, p.x, topYPos, this.w, this.h);

            // bottom pipe
            ctx.drawImage(sprite, this.bottom.sX, this.bottom.sY, this.w,
this.h, p.x, bottomYPos, this.w, this.h);
        }
    },

    update: function(){
        if(state.current !== state.game) return;

        if(frames%100 == 0){
            this.position.push({
                x : cvs.width,
                y : this.maxYPos * ( Math.random() + 1)
            });
        }
        for(let i = 0; i < this.position.length; i++){
            let p = this.position[i];

            let bottomPipeYPos = p.y + this.h + this.gap;
            // COLLISION DETECTION
            // TOP PIPE
            if(bird.x + bird.radius > p.x && bird.x - bird.radius < p.x + this.w && bird.y + bird.radius > p.y && bird.y - bird.radius < p.y + this.h){
                state.current = state.over;
                HIT.play();
            }
            // BOTTOM PIPE
            if(bird.x + bird.radius > p.x && bird.x - bird.radius < p.x + this.w && bird.y + bird.radius > bottomPipeYPos && bird.y - bird.radius < bottomPipeYPos + this.h){
                state.current = state.over;
                HIT.play();
            }

            // MOVE THE PIPES TO THE LEFT
            p.x -= this.dx;

            // if the pipes go beyond canvas, we delete them from the array
            if(p.x + this.w <= 0){
                this.position.shift();
                score.value += 1;
                SCORE_S.play();
                score.best = Math.max(score.value, score.best);
                localStorage.setItem("best", score.best);
            }
        }
    },

    reset : function(){
        this.position = [];
    }

}

// SCORE
const score= {
    best : parseInt(localStorage.getItem("best")) || 0,
    value : 0,

    draw : function(){
        ctx.fillStyle = "#FFF";
        ctx.strokeStyle = "#000";

        if(state.current == state.game){
            ctx.lineWidth = 2;
            ctx.font = "35px Teko";
            ctx.fillText(this.value, cvs.width/2, 50);
            ctx.strokeText(this.value, cvs.width/2, 50);

        }else if(state.current == state.over){
            // SCORE VALUE
            ctx.font = "25px Teko";
            ctx.fillText(this.value, 225, 186);
            ctx.strokeText(this.value, 225, 186);
            // BEST SCORE
            ctx.fillText(this.best, 225, 228);
            ctx.strokeText(this.best, 225, 228);
        }
    },

    reset : function(){
        this.value = 0;
    }
}

// DRAW
function draw(){
    ctx.fillStyle = "#70c5ce";
    ctx.fillRect(0, 0, cvs.width, cvs.height);

    bg.draw();
    pipes.draw();
    fg.draw();
    bird.draw();
    getReady.draw();
    gameOver.draw();
    score.draw();
}

// UPDATE
function update(){
    bird.update();
    fg.update();
    pipes.update();
}

// LOOP
function loop(){
    update();
    draw();
    frames++;
```

```
        requestAnimationFrame(loop);
    }
    loop();
```

The game comes from here: https://github.com/CodeExplainedRepo/Original-Flappy-bird-JavaScript (https://github.com/CodeExplainedRepo/Original-Flappy-bird-JavaScript)

But conceptually, there is nothing really in the language that you do not know from Python yet.

Almost, we are addressing the missing pieces today.

JavaScript is a programming language that is actually -besides the syntactic differences- quite similar to Python.

You have:

- basic datatypes, such as integers, floats, strings, lists, dictionaries (hashmaps) etc.
- basic syntactic structures, such as `for` -loops, `while` -loops, `if` - `else` conditionals, etc.
- functions and methods, classes, objects, and modules
- ...

That is, when learning JavaScript in the next semesters, you do not have to start learning programming from scratch. Instead, just try to map what you know from Python to how it is written in Python.

That is, it will be similar to learning Italian when you know already another Latin language, such as French.

Agenda

To easen your start with JavaScript in the next semesters, I would like to cover four concepts

- Concurrent programs
- First-class functions
- Event-driven programming
- Web-applications (Python in the Browser)

Running Code Concurrently

In [2]:

```python
import sys
import subprocess
from time import sleep


def say_something(msg, speak=True):
    if speak:
        cmd = f"""osascript -e 'say "{msg}"'"""
        subprocess.run(cmd, shell=True)
    else:
        for letter in msg:
            sys.stdout.write(letter)
            sys.stdout.flush()
            sleep(0.1)
        print()  # Finish with a newline
```

In [14]:

```python
msg = 'Hi, how are you? Do you have a nice day?'
say_something(msg, False)
```

```
Hi, how are you? Do you have a nice day?
```

In [5]:

```python
say_something(msg, speak=False)
```

```
Hi, how are you?
```

Reflection

Running code takes time. The longer your computation in the body of a function the longer this function is running.

While one function is running, no other function can be executed. That is, one statement **blocks** the execution of the next statement.

That is, all the statements in your program run strictly sequentially, so far:

In [15]:

```python
msg1 = 'Good morning! I am just telling you a very very long story!'
msg2 = 'Argh, I forgot, I wanted to tell you an even longer story!'

say_something(msg1)
say_something(msg2)
```

That is not always desired. Sometimes you want to execute code *concurrently*, i.e., at the same time.

For example, on your computer you want to listen to Spotify, while programming in Mu-editor, while downloading a book, while your email client is checking for new emails.

All these programs run concurrently on your computer and your operating system decides, which program is executed when for a tiny period of time.

In [16]:

```python
import threading


msg1 = 'Good morning! I am just telling you a very very long story!'
msg2 = 'Argh, I forgot, I wanted to tell you an even longer story!'


background_thread1 = threading.Thread(target=say_something, args=[msg1])
background_thread1.start()
background_thread2 = threading.Thread(target=say_something, args=[msg2])
background_thread2.start()
```

Recap: Sequential execution of code

In [17]:

```python
msg1 = 'Good morning! I am just telling you a very very long story!'
msg2 = 'Argh, I forgot, I wanted to tell you an even longer story!'


say_something(msg1, False)
say_something(msg2, False)
```

```
Good morning! I am just telling you a very very long story!
Argh, I forgot, I wanted to tell you an even longer story!
```

Recap: Concurrent execution of code

In [21]:

```
import threading


msg1 = 'Good morning! I am just telling you a very very long story!'
msg2 = 'Argh, I forgot, I wanted to tell you an even longer story!'


background_thread1 = threading.Thread(target=say_something, args=[msg1, False
background_thread1.start()
background_thread2 = threading.Thread(target=say_something, args=[msg2, False
background_thread2.start()
```

```
GAorgohd ,m oIr nfionrgg!ot ,I  Ia mw ajnustte d ttelol itnelg l yyo
uou a  avne reyv evn erlyon lgoerng s tstoorryy!!
```

Concurrency != Parallelism

Btw., what was that? A function name as an argument?

In [23]:

```
def say_something(msg, speak=True):
    if speak:
        cmd = f"""osascript -e 'say "{msg}"'"""
        subprocess.run(cmd, shell=True)
    else:
        for letter in msg:
            sys.stdout.write(letter)
            sys.stdout.flush()
            sleep(0.1)
        print()  # Finish with a newline


background_thread1 = threading.Thread(target=say_something, args=[msg1, False
background_thread1.start()
```

```
Good morning! I am just telling you a very very long story!
```

Your Turn!

- Write the following code in Mu-editor.
- Debug it by setting a breakpoint on line six.
- Inspect in the debugger window to the right, which variables are declared.
- What is the value of `say_something` there?
- What is `say_something` there?

In [ ]:

```python
def say_something(msg):
    print(msg)
    return 'I said: ' + msg


say_something(msg)
```

Your Turn!

- Now, do the same with the following program.
- What is the value of `say_something` there?
- What is `say_something` there?
- What is the value of `another_variable` there?
- What is `another_variable` there?
- What happens when you execute (Step over) line seven?
- What is printed?

In [ ]:

```python
1 def say_something(msg):
2     print(msg)
3     return 'I said: ' + msg
4
5
6 another_variable = say_something
7 result = another_variable('Uiuiui!')
8 print(result)
```

First-class functions

In Python as well as in JavaScript, functions are just variables pointing to their implementation. That is, their implementation is their value, which is first executed when the parenthesis appear.

[See an example the above example visualized here. (http://pythontutor.com/visualize.html#code=def%20say_something%28msg%29%3A%0A%20%20%20%2 frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)](http://pythontutor.com/visualize.html#code=def%20say_something%28msg%29%3A%0A%20%20%20%2)

In [ ]:

```python
background_thread1 = threading.Thread(target=say_something, args=[msg1, False
background_thread1.start()
```

However, do not use that to *rename* your functions! That is a bad coding practice:

```
def say_something(msg):
    print(msg)
    return 'I said: ' + msg


another_variable = say_something
result = another_variable('Uiuiui!')
print(result)
```

However, we can use this pattern to write programs that react on events, such as user input, time changes, etc.

Batch programming vs. Event-driven programming

Think about how you would write a program in which you want to steer a turtle over the screen with your current knowledge.

Very likely, your program would look similar to the code in `turtle_batch.py` , see it below. Run it from the command-line with `python turtle_batch.py`

In [ ]:

```python
import sys
import turtle


turtle.setup(400, 500)
window = turtle.Screen()
window.title('Handling keypresses!')
tess = turtle.Turtle()


def move():
    tess.forward(30)

def turn_left():
    tess.left(45)

def turn_right():
    tess.right(45)

def quit():
    window.bye()
    sys.exit()


while True:
    command = input('What shall I do? ')
    print('Received command: ' + command)
    if command == 'm':
        move()
    elif command == 'l':
        turn_left()
    elif command == 'r':
        turn_right()
    elif command == 'q':
        quit()
    else:
        print('Valid commands are `m`, `l`, `r`, `q`...')
```

Using the `input` function here, makes your program block until the user hits the return key on the command line after the corresponding command.

This is likely not what you really want or do your computer games or other applications behave like that?

Instead you rather like your program to execute the functions `move` , `turn_left` , `turn_right` , and `quit` whenever you hit a desired key.

Key presses are registered by your operating system, which creates an event for each key press and puts the event on an event-*queue*.

You can register your functions so that they are executed whenever a certain event appears on the event-queue.

See that for example in `turtle_event_driven.py` .

In [ ]:

```python
import sys
import turtle


turtle.setup(400, 500)
window = turtle.Screen()
window.title('Handling keypresses!')
tess = turtle.Turtle()

say_thread = 'I never said anything...'

# The next four functions are our 'event handlers'.
def go_up():
    tess.forward(30)


def turn_left():
    tess.left(45)


def turn_right():
    tess.right(45)


def quit():
    window.bye()  # Close down the turtle window


# These lines 'wire up' keypresses to the handlers we've defined.
window.onkey(go_up, 'Up')
window.onkey(turn_left, 'Left')
window.onkey(turn_right, 'Right')
window.onkey(quit, 'q')


# Now we need to tell the window to start listening for events,
# If any of the keys that we're monitoring is pressed, its
# handler will be called.
window.listen()
window.mainloop()
```

There, you connect you functions to certain key events:

```
window.onkey(go_up, 'Up')
window.onkey(turn_left, 'Left')
window.onkey(turn_right, 'Right')
window.onkey(quit, 'q')
```

Here, you have to use the pointers to the functions as values. Otherwise you would execute the respective function directly.

And you listen to and start the event-loop:

```
window.listen()
window.mainloop()
```

Event-loop??? Meeting the Queue again.

```
while True:
    # Get the next event from the operating system
    event = get_next_event()

    # Get the function that is assigned to handle this event
    a_function_to_handle_the_event = event-handlers[event]

    # If a function has been assigned to handle this event,
    # call the function
    if a_function_to_handle_the_event:
        # Call the event-handler function
        a_function_to_handle_the_event()

    # Stop processing events if the user gives a command to
    # stop the application
    if window_needs_to_close:
        break  # out of the event-loop
```

Read more on GUI-programming and the event-loop in TKinter programs here (https://runestone.academy/runestone/books/published/thinkcspy/GUIandEventDrivenProgramming/toctree.

Obviously, `event-handlers[event]` return functions, which are assigned to a variable `a_function_to_handle_the_event`. The respective function is called with `a_function_to_handle_the_event()`

Timer Events - Repetitive Tasks with a Timer

Now, let's rewrite the previous program in which you steer Tess the turtle to advance automatically every so and so many seconds and not any longer when a certain key is pressed.

For that, we register the function `go_up` to be triggered by a timer event.

In [ ]:

```python
import sys
import turtle


turtle.setup(400, 500)
window = turtle.Screen()
window.title('Handling keypresses!')
tess = turtle.Turtle()


def go_up():
    tess.forward(30)
    window.ontimer(go_up, 160)


def turn_left():
    tess.left(45)


def turn_right():
    tess.right(45)


def quit():
    window.bye()
    sys.exit()


# These lines 'wire up' keypresses to the handlers we've defined.
window.onkey(turn_left, 'Left')
window.onkey(turn_right, 'Right')
window.onkey(quit, 'q')

window.ontimer(go_up, 160)

window.listen()
window.mainloop()
```

Long running functions block the UI thread

Let's say something on a key event for speaking.

What happens?

Can you observe the queue somewhere?

In [ ]:

```python
import turtle
import platform
import threading
import subprocess
from time import sleep


turtle.setup(400, 500)
window = turtle.Screen()
window.title('Handling keypresses!')
tess = turtle.Turtle()


# The next four functions are our "event handlers".
def go_up():
    tess.forward(30)
    window.ontimer(go_up, 160)


def turn_left():
    tess.left(45)


def turn_right():
    tess.right(45)


def quit():
    window.bye()


def say_something(msg, speak=True):
    if speak:
        cmd = f"""osascript -e 'say "{msg}"'"""
        subprocess.run(cmd, shell=True)
    else:
        for letter in msg:
            sys.stdout.write(letter)
            sys.stdout.flush()
            sleep(0.1)
        print()  # Finish with a newline


def speak():
    msg = 'Move Tess the turtle!'
    say_something(msg, speak=True)


# These lines 'wire up' keypresses to the handlers we've defined.
window.onkey(turn_left, 'Left')
window.onkey(turn_right, 'Right')
window.onkey(quit, 'q')
window.onkey(speak, 's')

window.ontimer(go_up, 160)


# Now we need to tell the window to start listening for events,
# If any of the keys that we're monitoring is pressed, its
# handler will be called.
window.listen()
window.mainloop()
```

In [ ]:

```
http://10.28.40.143:8000
```

Python in the Browser

Web-applications are usually written in JavaScript in combination with HTML and CSS.

With Brython (https://brython.info) you can write web-applications in Python too.

Brython is a Python interpreter written in JavaScript, which can run in the browser.

```
<html>
  <head>
    <script type="text/javascript"
       src="https://cdnjs.cloudflare.com/ajax/libs/brython/3.7.1/brython.min.js">
    </script>
    <script type="text/javascript"
    src="https://cdnjs.cloudflare.com/ajax/libs/brython/3.7.1/brython_stdlib.js">
</script>
    <script type="text/python" src="./clicker.py"></script>
  </head>

  <body onload="brython()">
    <canvas id="playground" width="400" height="400" style="border-color:#000;border-style:solid;border-width:1px;margin-left:5em;"></canvas>
    <br>
    <button id="turnbutton">Turn!</button>
    <span id="score"></span>

  </body>

</html>
```

```
from browser import document, timer, alert


class GameState:
    def __init__(self):
        self.x = 10
        self.y = 10
        self.x_step = 5
        self.y_step = 5

        self.score = 0


PLAYER_SIZE = 20

canvas = document['playground']
ctx = canvas.getContext('2d')
ctx.fillStyle = 'green'

clicker_state = GameState()
```

```
def restart_game():
    alert('Banging against the wall?')
    clicker_state.x = clicker_state.y = 10
    clicker_state.x_step = clicker_state.y_step = 5
    clicker_state.score = 0


def clear_screen():
    ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height)


def draw():
    clear_screen()

    clicker_state.x += clicker_state.x_step
    clicker_state.y += clicker_state.y_step

    if ((clicker_state.x < 0) or
       (clicker_state.x > ctx.canvas.width - PLAYER_SIZE) or
       (clicker_state.y < 0) or
       (clicker_state.y > ctx.canvas.height - PLAYER_SIZE)):
        restart_game()

    ctx.fillRect(clicker_state.x, clicker_state.y, PLAYER_SIZE, PLAYER_SIZE)
    clicker_state.score += 1
    document['score'].textContent = 'Your score: ' + str(clicker_state.score)


def turn_it():
    going_right = (clicker_state.x_step > 0)
    going_left = (clicker_state.x_step < 0)
    going_down = (clicker_state.y_step > 0)
    going_up = (clicker_state.y_step < 0)
    if (going_right and going_down) or (going_left and going_up):
        clicker_state.y_step *= -1
    elif (going_right and going_up) or (going_left and going_down):
        clicker_state.x_step *= -1

    clicker_state.score -= 20


timer.set_interval(draw, 50)

document['turnbutton'].bind('click', turn_it)
```

Workshop

1. Download the clicker game from the folder `clicker_game` under `session-10` . It consists of two files, `index.html` and `clicker.py` and store them in on directory `clicker_game` on your computer.

   - Start the game on your computer using the following commands:

   ```
   $ cd clicker_game
   $ python -m http.server
   ```

   This starts a webserver on your computer in this local directory.

   - Play the game locally by pointing your browser to [http://localhost:8000/ (http://localhost:8000/)](http://localhost:8000/)
   - You can always stop the web-server in the terminal by pressing `ctrl+c` , i.e., press the `ctrl` and the `c` keys simultaneously to stop the server.

Finding your own IP address

- Find the IP address of your computer by running the following small Python program, which will print your computer's IP address:

```
import socket


host_name = socket.gethostname()
ip_address = socket.gethostbyname(host_name)
print(f"My computer's IP address: {ip_address}")
```

- Alternatively, you can run either `ifconfig` on (MacOS/Linux) or `ipconfig` on Windows to find the IP address of your computer. On MacOS it is usually listed behind the entry `inet` below the networkcard, which is ususally `en1` . See for example, for [MacOS (https://www.youtube.com/watch?v=CTA3hu7XG8k)](https://www.youtube.com/watch?v=CTA3hu7XG8k) or for [Windows (https://www.youtube.com/watch?v=PZtbGoTaN3E)](https://www.youtube.com/watch?v=PZtbGoTaN3E)
- Note down the IP address of your computer and share it with your group mates.
- Now, start the webserver again via:

```
$ cd clicker_game
$ python -m http.server
```

- Instead of playing the game on your computer, let your group mates navigate in their browsers to http://:8000/
- Now, they are playing the game with the code that was downloaded from your computer.

OBS: To be able to connect a browser on another computer to the given IP address, all involved devices, i.e., the computer serving the game, and all the clients have to be on the same network. At ITU this is for example the `ITU++` network.

Extending the Clicker game

Choose to implement either of the following:

A) Adding levels to the game

Add another field `Your level:` next to the field `Your score:` on the game page. You should start playing in level `0` and every `100` steps the level should increase.

Hint:

- Add another property `steps` to the class `GameState` and increase it in the `draw` function similar to how `clicker_state.score += 1` is increased on line 48.
- Add another property `level` to the class `GameState` and increase it in the `draw` function similar to how `clicker_state.score += 1` is increased on line 48 whenever `steps` another multiply of `100`.
- Additionally, to modifying the code in `clicker.py` you have to modify the code in `index.html`
  - You have to add another `span` element like `<span id="level"></span>` on line 17 in `index.html`.
  - To write to it you need an extra line in `clicker.py`, which is similar to: `document['score'].textContent = 'Your score: ' + str(clicker_state.score)`

B) Adding a chaos mode

Add another button to the game page by adding the following line to `index.html`:

```
<button id="chaosbutton">Chaos!</button>
```

Create a new function `chaos` in `clicker.py` and add a binding to it in the very button of `clicker.py`, which looks like:

```
document['chaos'].bind('click', chaos)
```

Lines 56 to 59 in `clicker.py` control in which direction the square turns whenever the `turn` button is clicked.

```
    if (going_right and going_down) or (going_left and going_up):
        clicker_state.y_step *= -1
    elif (going_right and going_up) or (going_left and going_down):
        clicker_state.x_step *= -1
```

Replace the two occurrences of `-1` with values (`chaos_x` and `chaos_y`) stored in the `GameState`. Let these values change randomly whenever the `chaos` button is clicked. Use for example:

```
clicker_state.chaos_x = random.choice([1, -1])
clicker_state.chaos_y = random.choice([1, -1])
```

in it.

C) Adding edible gems

Place every now and then (every so and so many steps) a red square on a random place on the game board.

Whenever your green square 'hits' the red square, the red square should disappear and the player should get a certain amount of extra points.