# Day 3: Controlling Playback and Exiting Video

## Outcomes:

- Understand how keyboard input is captured in OpenCV
- Know what `cv2.waitKey()` does and what it returns
- Understand ASCII values and why keys are represented as numbers
- Understand bitwise AND (&) and why 0xFF is used
- Implement a keyboard-controlled exit from a video loop

## What is the need for time control?

A computer can display frames at a much faster rate than the real video speed.

Without time control:

- the window may not update correctly
- key presses won't be detected
- the program may freeze or close instantly

OpenCV uses `waitkey(delay)` to solve this

Consider the line:

```
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

This single line controls both delay and exit-logic.

## The `waitkey()` function

Purpose:

- Wait for a specific time in milliseconds (ms)
- Checks if a key was pressed during that time

Return values:

- Return type: `int`
- If no key is pressed it returns `-1`
- If a key is pressed, returns a number representing that key
    - For example, if q is pressed, it returns `‘113’` as it is the ASCII / Unicode for the key `‘q’`.

In short, `cv2.waitKey()` returns an integer representing the key pressed, or `-1` if no key is pressed.

## `0xFF`

Breakdown of `0xFF`:

- `0x` – represents that the number is a hexadecimal number
- `FF` –hexadecimal representation equivalent to `1111 1111` in binary

Note: Here, ‘&’ is a bitwise AND operator.

`waitkey()` function returns a 32-bit number, in which only the last 8 bits represent the key, hence we need to extract these last 8 bits out of the whole.

The purpose of using `0xFF` here is to extract last 8-bit number from those 32-bits returned by the `waitkey()` function.

### Example

Suppose that a key is pressed, and `waitkey()` returns

`00000000 00000000 00000000 01110001`

`waitkey() & 0xFF` (which represents `1111 1111`), bitwise AND operation helps extract the last 8 bits of data from the 32-bit number.

`00000000 00000000 00000000 01110001 & 11111111` gives `01110001`,

which is the exact 8-bit number from the right of the 32-bit number, as `0 & 1` gives `0` and `1 & 1` gives `1` itself.

## The `ord()` function

`ord()` is a built-in python function.

Purpose:

- Converts a single character into its ASCII (or Unicode) numeric value.
- Input: a string of length 1 (eg. `‘q’, ‘A’, ‘7’`)
- Output: Integer representing that character

`ord(‘q’)` converts the character `‘q’` into a number

### Example

```
ord('q')   # 113
ord('A')   # 65
ord('0')   # 48
```

- Since python stores every number in binary, `113` is converted into binary
- then compared with the 8-bit number returned after the bitwise AND operation between the 32-bit number and `0xFF` (`1111 1111`)
- if both are same, the loop is broken and the program stops.