# Introduction to Programming 2025-26847

## Lab 2 - Creació d'un joc de Pong

Aquest laboratori s'ha de fer en grups de 2 estudiants i s'ha de lliurar a través d'Aula Global.

El fitxer a lliurar serà un fitxer de Python (.py) anomenat **PNNN_TNN_P2_U1_U2.py** (on **PNNN** és el grup de lab amb 3 xifres — p. ex. P101 —, **TNN** és el número d'equip — p. ex. T18 —, **P2** indica el laboratori (aquest és el primer) i **U1** i **U2** són els identificadors UXXXXXX de cada membre del grup — el codi U es pot trobar al revers del carnet d'estudiant).

També heu de lliurar un **informe escrit** que comenti la vostra implementació i expliqui els aspectes que considereu més importants. Aquest informe hauria de ser preferentment en format pdf, amb una extensió recomanada de 2–4 pàgines.

**Com que Aula Global no permet pujar 2 fitxers, pugeu un sol fitxer zip que contingui tots dos** (.py i .pdf).

## 1. Introducció

En aquest laboratori, practicarem **definir i utilitzar funcions** desenvolupant una versió simplificada del joc clàssic **Pong**. L'objectiu no és construir un joc complex, sinó organitzar el codi en funcions clares i reutilitzables i entendre com interactuen per controlar la lògica i el comportament del joc.

A diferència del **Lab 1**, que es va desenvolupar i lliurar utilitzant un **notebook de Google Colab**, aquest lab es durà a terme amb una **instal·lació local de Python** i l'entorn de desenvolupament integrat **VS Code** (IDE). Treballar localment us permetrà familiaritzar-vos amb com s'organitzen i s'executen projectes de Python fora de Colab, així com amb l'execució i depuració de fitxers `.py` directament.

Abans de començar aquest lab, assegureu-vos d'haver instal·lat correctament **Python** i **VS Code** seguint la guia pas a pas del document **[Instal·lació de Python i VS Code]**. Aquest document explica com preparar l'entorn perquè pugueu editar i executar fitxers de Python des del vostre ordinador.

### 1.2 Objectius d'aprenentatge

Al final d'aquest lab, hauríeu de ser capaços de:

- **Instal·lar i configurar** un entorn local de Python i l'**IDE VS Code**, i entendre com **obrir, editar i executar** fitxers `.py`.
- Entendre què és una **llibreria de Python** i com **instal·lar paquets externs** (com `pygame`) utilitzant `pip`.
- **Definir, cridar i combinar funcions** per estructurar un programa de forma lògica i modular — fent un ús correcte de **paràmetres i valors de retorn**, i **evitant dependre de variables globals**.
- Aplicar estructures de control (`if`, `while`, etc.) dins de funcions per implementar lògica de joc senzilla.
- Entendre el flux bàsic d'un **bucle de joc**, i com utilitzar funcions per gestionar i actualitzar l'estat del joc.
- Practicar l'escriptura de **codi net i llegible** que segueixi les especificacions donades i eviti característiques innecessàries com llistes, diccionaris o classes.

---

### 1.3 Notes importants

L'ús de **variables globals**, **llistes**, **diccionaris** i **classes** està **estrictament prohibit** en aquest lab. Totes les tasques s'han d'implementar utilitzant únicament variables simples, paràmetres de funció i valors de retorn.

Qualsevol forma de **plagi** o **ús cec o irreflexiu d'eines d'IA** serà **severament penalitzada**. S'espera que els estudiants entenguin cada part del codi que entreguen. El professorat us pot demanar que **expliqueu seccions específiques del vostre programa** per verificar-ne la comprensió.

---

### 1.4 Ús de llibreries externes en aquest lab

En aquest lab farem servir una **llibreria externa de Python** anomenada **`pygame`**, que proporciona les eines necessàries per dibuixar, animar i interactuar amb elements gràfics senzills. Abans d'instal·lar `pygame`, assegureu-vos d'haver completat la configuració descrita al document **Instal·lació de Python i VS Code**, i que tant **Python** com **VS Code** estan correctament instal·lats i funcionen al vostre ordinador. Un cop l'entorn estigui a punt, seguiu les instruccions següents per instal·lar i provar `pygame`.

## 2. Instal·lació de `pygame` i què són les llibreries de Python

### 2.1 Què són les llibreries?

Una **llibreria** (o **mòdul**) a Python és una *col·lecció de codi preparat* que podeu reutilitzar en lloc d'escriure-ho tot des de zero. Per exemple, `pygame` és una llibreria que proporciona eines per dibuixar formes, gestionar el teclat, reproduir sons i controlar un bucle de joc.

Podeu pensar en les llibreries com a **caixes d'eines**: cadascuna afegeix capacitats específiques a Python (gràfics, matemàtiques, anàlisi de dades, accés web, etc.).

Python ja inclou moltes llibreries estàndard (com `math`, `random` o `time`), però també podeu instal·lar-ne d'*externes* creades per tercers — com `pygame`.

---

### 2.2 Què és `pip`?

`pip` és el **gestor de paquets** de Python — una eina de línia d'ordres que descarrega i instal·la llibreries del repositori oficial en línia anomenat **PyPI** (Python Package Index).

S'utilitza des del terminal (Windows, macOS o Linux). Per exemple:

```
pip install pygame
```

Aquesta ordre indica a `pip` que:

1. es connecti a PyPI,
2. descarregui l'última versió de la llibreria `pygame`,
3. i la instal·li al vostre entorn local de Python.

Un cop instal·lada, la podreu importar en qualsevol fitxer `.py`:

```
import pygame
```

---

### 2.3 Nota important per a usuaris de macOS

A macOS, l'ordre `python` sol referir-se a una versió antiga (2.x). Els sistemes moderns utilitzen **`python3`** i **`pip3`**.

Per tant, sempre que vegeu una ordre com:

```
python ...
```

normalment hauríeu d'escriure en canvi:

```
python3 ...
```

i de manera similar:

```
pip3 install pygame
```

> Consell: Podeu comprovar quina ordre funciona escrivint `python --version` o `python3 --version` al terminal.

---

### 2.4 Com instal·lar `pygame` (pas a pas)

**Windows / macOS / Linux (a VS Code o al terminal):**

1. Obriu el terminal (o el terminal de VS Code: *View → Terminal*).
2. Escriviu una de les opcions següents, segons el vostre sistema:

   **Windows / Linux:**

   ```
   pip install pygame
   ```

   **macOS:**

   ```
   pip3 install pygame
   ```

3. Premeu Enter.
4. Espereu fins que vegeu un missatge que acabi amb: `Successfully installed pygame-...`

---

### 2.5 Prova de la instal·lació

Per confirmar que `pygame` funciona correctament, executeu aquesta ordre:

```
python3 -m pygame.examples.aliens   # macOS / Linux
```

o bé

```
python -m pygame.examples.aliens    # Windows
```

Si s'obre un petit joc d'estil arcade, tot està correctament instal·lat!

## 3. El joc del Pong - Construeix el teu propi Pong

**Pong** és un dels primers i més icònics videojocs mai creats. Creat per **Atari el 1972**, simula un partit senzill de tennis de taula, on dues pales (paddles) colpegen una pilota en moviment d'un costat a l'altre de la pantalla. Cada jugador mou la seva pala verticalment per evitar que la pilota surti pel seu costat i intenta que el contrari falli — sumant un punt quan això passa.

En aquest lab, **recreareu una versió simplificada de Pong amb Python**. L'objectiu no és construir un joc complex, sinó **practicar el disseny i l'ús de funcions**, organitzar el codi amb claredat, i entendre com els paràmetres, els bucles i la lògica condicional treballen plegats per produir comportaments.

Partireu d'un **marc (fitxer plantilla) en Python** que us proporcionem. Aquest fitxer ja inclou l'estructura del joc i diverses constants i capçaleres de funció predefinides. La vostra tasca és **completar o implementar les funcions que falten**, seguint els docstrings i comentaris facilitats. Cada funció se centra en un aspecte petit i ben definit del joc (moviment, col·lisions, gestió d'entrada, etc.), permetent reconstruir progressivament tota la lògica.

Al final del lab, podreu jugar la vostra pròpia versió de *Pong* — descendent directe del joc que ho va començar tot.

**Lectures addicionals:**

- Viquipèdia – Pong
- Juga'n una versió web

## 4. Treballar amb Visual Studio Code (VS Code)

Per a aquest lab, farem servir **VS Code** com a entorn de programació.

Si no heu utilitzat mai VS Code, no patiu — només necessitareu unes poques operacions bàsiques per completar aquest lab.

---

### 4.1 Obrir el fitxer

- Inicieu VS Code.
- Des del menú superior, trieu **File → Open File...**, i seleccioneu el fitxer proporcionat `lab2_pong_template_CAT.py`.
- El codi apareixerà a la finestra central de l'editor.
- També podeu **fer doble clic al fitxer** des de l'explorador de fitxers de l'esquerra per obrir-lo.

---

### 4.2 Executar el programa

Hi ha **dues maneres** d'executar el programa:

➤ Opció A: des de la icona "Run"

- A l'angle superior dret de l'editor, cliqueu la **icona del triangle verd ("Run")**.
- VS Code obrirà automàticament un terminal a la part inferior i executarà el fitxer per vosaltres.

➤ Opció B: des del terminal

- Deseu qualsevol modificació (`Ctrl+S` o `⌘S` a macOS).
- Obriu un terminal manualment: **View → Terminal**.
- Al terminal, escriviu:

  ```
  python lab2_pong_template_CAT.py
  ```

  o, a macOS:

  ```
  python3 lab2_pong_template_CAT.py
  ```

Si tot funciona, s'obrirà una finestra nova amb el joc.

---

### 4.3 Entendre i navegar pels missatges d'error

Si alguna cosa falla, Python mostrarà un **missatge d'error** al terminal. Cada missatge inclou:

- El **nom del fitxer i el número de línia** on s'ha produït l'error.
- El **tipus d'error** (per exemple, `SyntaxError`, `NameError`, `TypeError`).
- Una breu **descripció** del que ha anat malament.

No patiu — els errors són completament normals! Llegiu el missatge amb atenció i utilitzeu-lo com a pista:

- Falta algun parèntesi?

- Heu escrit malament el nom d'alguna variable?
- Heu oblidat retornar algun valor?

**Consell:** A VS Code, podeu **Maj + clic** en un missatge d'error del terminal per saltar **directament a aquella línia** del codi. Això fa que solucionar problemes sigui molt més ràpid.

## 5. El fitxer de Python proporcionat

### 5.1 Omple les funcions sol·licitades

Se us proporciona `lab2_pong_template_CAT.py`, un **fitxer plantilla de Python** que conté l'estructura del joc i diverses funcions, constants i comentaris predefinits.

Aquest fitxer actua com un **marc** o **esquelet** que anireu completant progressivament. Algunes funcions ja estan escrites, mentre que d'altres només contenen un **docstring** i un comentari `# TODO` — aquestes són les parts que heu d'implementar.

Els docstrings expliquen clarament:

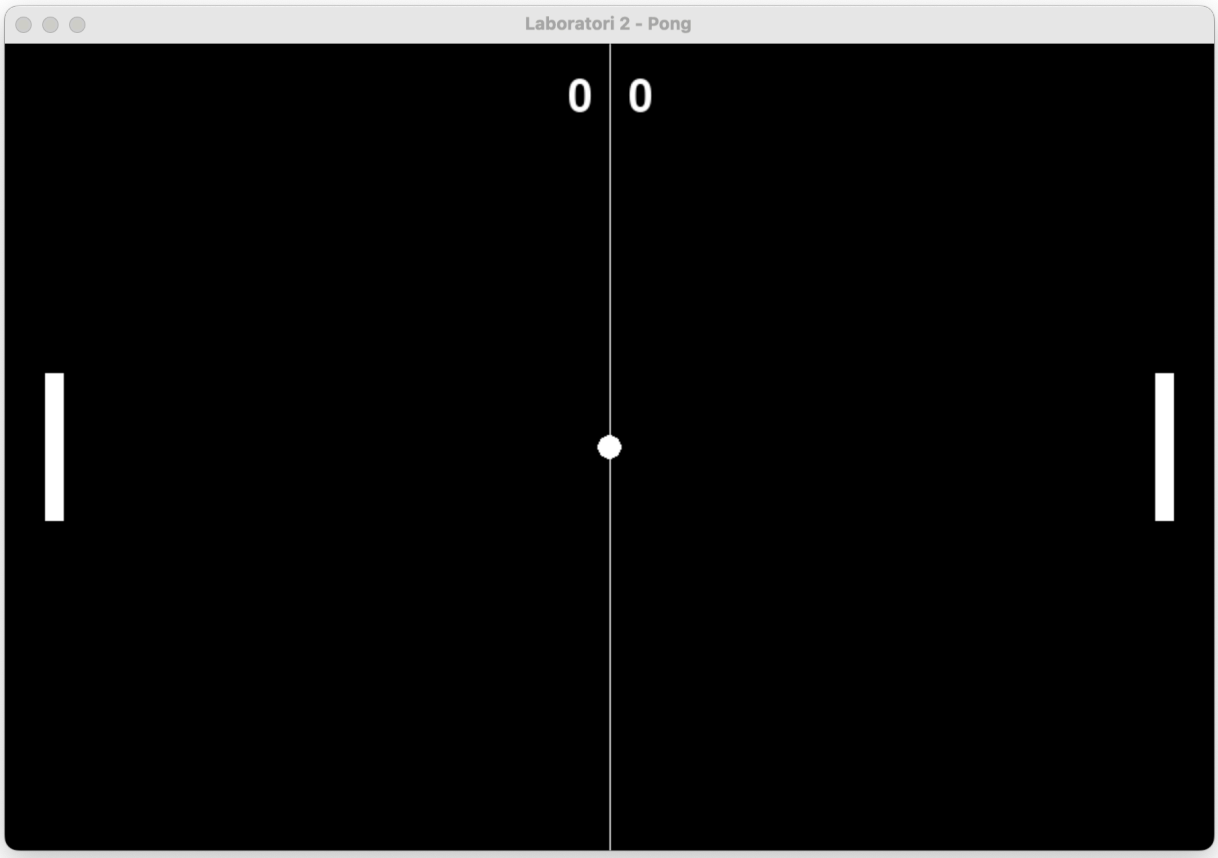
- què ha de fer cada funció,
- quins paràmetres rep, i
- què ha de retornar.

En alguns casos, també s'inclouen exemples o pistes per guiar el vostre raonament.

**Important:** No heu de canviar els noms de les funcions ni els seus paràmetres, ja que altres parts del programa hi depenen. Centreu-vos a completar la lògica interna de cada funció, respectant les especificacions.

Aquest disseny us permet **executar i provar** el joc fins i tot abans d'acabar-lo del tot.

De fet, si executeu el fitxer proporcionat **sense cap modificació**, ja s'obrirà la finestra del joc (pales, pilota, marcador...) — però **res no es mourà encara**. Haureu de començar a implementar funcions per fer que el joc sigui jugable de manera progressiva.




---

### 5.2 Ordre recomanat d'implementació de les funcions

Funcions principals del joc

1. `move_paddle()`
    - Calcula el nou `y` a partir de `y` actual, `direction` i `speed`.
    - Mantén la pala completament dins la pantalla (part superior ≥ 0, part inferior ≤ alçada de la finestra).
2. `move_ball()`

- Actualitza `x, y` utilitzant els `vx, vy` actuals.
- Aquí no hi ha col·lisions — només moviment pur.

3. `collide_ball_walls()`
   - Si la pilota toca la part superior/inferior, inverteix `vy` i empeny-la una mica dins dels límits.
   - **No** puntuïs aquí (només rebot).
4. `collide_ball_paddle()`
   - Detecta el solapament entre la caixa de la pilota i el rectangle d'una pala (AABB).
   - Si hi ha impacte, inverteix `vx`; opcionalment ajusta `vy` segons el punt d'impacte; empeny la pilota just fora de la pala per evitar que "s'enganxi".
5. `reset_ball()`
   - Torna a centrar la pilota i estableix `vx, vy` inicials (la direcció depèn de `to_right`).
   - Opcional: una petita aleatorietat a `vy` perquè els serveis variïn.

Consell: Després de cada funció, **executa el joc** i observa un nou comportament. Corregeix problemes aviat.

Funcions de configuració

6. `ask_yes_no()`
   - Repeteix fins que l'usuari respongui `y/n` (o `yes/no`). Retorna `True/False`.
7. `read_int_in_range()`
   - Demana un enter dins de `[low, high]`. Torna a demanar en cas d'entrada invàlida.
8. `get_config()`
   - Pregunta si es vol obrir un menú de configuració.
   - Si és que sí, llegeix: alçada de la pala, velocitat de la pala, velocitat X de la pilota, velocitat Y de la pilota, radi de la pilota (en aquest ordre).
   - Retorna aquests cinc valors (no modifiquis constants globalment).
9. `change_colors()`
   - Pregunta si es volen canviar els colors de fons/primer pla.
   - Si és que sí, crida `read_color()` dues vegades i retorna `(bg, fg)`.
10. `read_color()`

- Demana enters `R`, `G`, `B` (0–255) i retorna'ls com un color.

Consell: Mantén la lògica de configuració **fora** del bucle del joc. Utilitza retorns, no globals.

---

Funcions opcionals (en qualsevol ordre)

11. `read_int_in_range_w_default()`

- Com `read_int_in_range()`, però **prement ENTER** retorna un valor per defecte.

12. `compute_ai_direction()`

- Retorna `-1/0/+1` per a un mode senzill de **Jugar contra la màquina** (la pala segueix la pilota amb una petita tolerància per evitar tremolors).

13. `show_message()`

- Estudia/modifica aquest ajudant per mostrar missatges personalitzats en pantalla (p. ex., "Paused", "Goal!", "Press SPACE to start").

---

### 5.3 Comprovacions ràpides a mesura que avances

- Després de `move_paddle()`: les pales es mouen però mai surten de la pantalla.
- Després de `move_ball()`: la pilota es desplaça com s'espera.
- Després de `collide_ball_walls()`: la pilota rebota a la part superior/inferior.
- Després de `collide_ball_paddle()`: la pilota rebota a les pales (sense enganxar-se).
- Després de `reset_ball()`: els gols tornen a posar la pilota al centre per al servei.
- Després de les funcions de configuració: les preguntes funcionen; els valors triats afecten clarament la jugabilitat.

Mantén els canvis petits, prova sovint, i llegeix els missatges d'error amb atenció — són pistes, no enemics.

Quan calgui, pots utilitzar `print()` dins de les funcions que hagis d'omplir/implementar, per traçar els valors dels paràmetres rebuts, comprovar càlculs, etc. Elimina aquests `print()` quan la funció funcioni!!!

## 6. Característiques opcionals

Implementar alguns d'aquests elements de **bonus** pot augmentar la nota del Lab 2 **fins a 12/10** (és a dir, **+2 punts extra com a màxim**). Aquests punts extra es comptaran a la mitjana final dels Labs (**(Lab1 + Lab2 + Lab3) / 3**).

1. `read_int_in_range_w_default()` Com `read_int_in_range()`, però si l'usuari prem **ENTER**, la funció retorna immediatament un **valor per defecte** (mostrat al prompt).
   - Focus: gestió d'entrada, valors opcionals per defecte, flux de control net.
2. `compute_ai_direction()` *(Jugar contra la màquina)* Retorna **-1 / 0 / +1** per moure una pala d'IA: amunt, quiet o avall.
   - Mantén-ho simple: la pala segueix la posició vertical de la pilota amb una petita **tolerància** per evitar tremolors.
3. `show_message()` Estudia/modifica l'ajudant proporcionat per mostrar **missatges personalitzats en pantalla** (p. ex., "Paused", "Goal!", "Press SPACE to start").
   - Focus: paràmetres clars de funció (screen, text), centrat del text, petites pauses.

---

### 6.2 "Jugar contra la màquina" — una estratègia senzilla

Objectiu: Substituir l'entrada de tecles del jugador dret per una petita funció que **segueixi la pilota**.

Idea

- Calcula el **centre de la pala**: `center = paddle_y + paddle_h / 2`.
- Compara `ball_y` amb `center`.
- Retorna:
  - `−1` si la pilota és **per sobre** del centre (mou cap amunt),
  - `+1` si la pilota és **per sota** del centre (mou cap avall),
  - `0` si la pilota és **prou a prop** (dins d'una tolerància), per evitar tremolors.

Ajustos que pots provar

- Augmentar la `tolerance` (p. ex., 10–12) ⟶ IA més suau i "mandrosa".
- Reduir-la (p. ex., 2–3) ⟶ més reactiva però pot tremolar.
- Limitar la velocitat global de la pala amb `SPEED_PADDLE` per mantenir el partit equilibrat.

## 7. El sistema de coordenades de Pygame i la detecció de col·lisions

### 7.1 Introducció

Pygame (com la majoria de sistemes gràfics 2D) utilitza una **graella cartesiana**, on cada posició a la pantalla es descriu amb una coordenada **(x, y)**. L'**origen (0, 0)** es troba a la **cantonada superior esquerra** de la finestra.

- L'**eix x** creix cap a la **dreta**.
- L'**eix y** creix **cap avall**.

Això vol dir:

- Moure un objecte *amunt* ⟶ decrementar `y`.
- Moure un objecte *avall* ⟶ incrementar `y`.
- La cantonada inferior dreta de la finestra té coordenades aproximadament `(WIDTH, HEIGHT)`.

Exemple (per a una finestra de 640×480):

```
(0,0) ──────────────► x
  │
  │
  ▼ y
```

- Cantonada superior esquerra ⟶ `(0, 0)`
- Centre de la pantalla ⟶ `(WIDTH/2, HEIGHT/2)`
- Cantonada inferior dreta ⟶ `(WIDTH, HEIGHT)`

---

### 7.2 Punts de referència en aquest projecte

- **Pales (paddles)** La posició de cada pala es defineix per les **coordenades de la seva cantonada superior esquerra**: `(paddle_x, paddle_y)` A partir d'aquí, la pala s'estén cap a la dreta segons l'amplada i cap avall segons l'alçada.
- **Pilota** La posició de la pilota `(ball_x, ball_y)` representa el **centre** de la pilota. El seu **radi** determina quant s'estén en totes direccions. Per tant, la "caixa contenidora" de la pilota va de `(ball_x − radius, ball_y − radius)` a `(ball_x + radius, ball_y + radius)`.

Aquesta distinció (centre vs. cantonada superior esquerra) és important per a la detecció de col·lisions i el dibuix.

Vegeu també: Documentació oficial de Pygame Rect

---

### 7.3 Col·lisions AABB — *Axis-Aligned Bounding Box*

Per detectar col·lisions, utilitzem un mètode geomètric senzill anomenat **AABB** (*Axis-Aligned Bounding Box*).

En lloc de treballar amb cercles o formes girades, tractem cada objecte (pilota i pales) com a **rectangles amb arestes paral·leles als eixos x i y**.

A la nostra implementació de Pong:

- Cada **rectangle de la pala** comença a `(paddle_x, paddle_y)` (la seva **cantonada superior esquerra**), amb amplada `PADDLE_W` i alçada `PADDLE_H`.
- El **rectangle de la pilota** està centrat a `(ball_x, ball_y)`, i s'estén `radius` píxels en totes direccions — de manera que amplada i alçada són `2 * BALL_RADIUS`.

Dos rectangles se solapen si s'intersecten en tots dos eixos:

```
Solapament horitzontal:  Solapament vertical:
paddle_x < ball_x + w    paddle_y < ball_y + h
and                      and
ball_x < paddle_x + W    ball_y < paddle_y + H
```

#### Significat dels símbols

| Símbol | Es refereix a | Significat |
|---|---|---|
| **paddle_x**, **paddle_y** | **Cantonada superior esquerra** de la pala | Posició inicial de la pala |
| **ball_x**, **ball_y** | **Cantonada sup. esq. de la caixa** de la pilota | Derivada del centre: (ball_x - radius, ball_y - radius) |
| **W** | **Amplada** de la pala | Mida horitzontal de la pala |
| **H** | **Alçada** de la pala | Mida vertical de la pala |
| **w** | **Amplada** de la pilota = `2 * radius` | Mida horitzontal de la pilota |
| **h** | **Alçada** de la pilota = `2 * radius` | Mida vertical de la pilota |

Si el solapament horitzontal i el vertical són certs ⟶ els rectangles s'intersecten ⟶ **hi ha col·lisió**.

## 8. Colors RGB a Pygame

Pygame representa els colors utilitzant **tuples RGB**, on cada color es defineix per tres components enters: `(R, G, B)` ⟶ **Vermell (Red), Verd (Green), Blau (Blue)**

Cada component és un enter entre **0 i 255**, on:

- `0` significa cap intensitat (gens d'aquell color),
- `255` significa intensitat màxima.

Per exemple:

```
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED   = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE  = (0, 0, 255)
GREY  = (128, 128, 128)
```

També podeu crear **colors personalitzats** barrejant valors diferents:

```
ORANGE = (255, 165, 0)
PURPLE = (160, 32, 240)
```

A Pygame, aquestes tuples RGB s'utilitzen a totes les funcions de dibuix — per exemple:

```
screen.fill(BLACK)  # omple el fons de negre
pygame.draw.rect(screen, WHITE, (x, y, width, height))  # rectangle blanc
pygame.draw.circle(screen, RED, (ball_x, ball_y), BALL_RADIUS)  # pilota vermella
```

Com que els colors són tuples senzilles, és fàcil permetre que l'usuari els triï llegint tres enters (R, G, B) i retornant-los plegats, tal com es fa a la funció `read_color()`.

Més informació: Documentació de dibuix de Pygame