# Flappy Bird Extensions!

## Extension 1: Printing Flappy Bird!

We can print words to the screen as well as displaying images! Let's print the words Flappy Bird to the screen so the user can't forget what game they're playing.

**Extension 1.1**

1) Store what colour you'd like to use in a variable called `colour`. You'll need to use the RGB values.
2) Store what size you'd like to use in a variable called `font`.
3) Store where you'd like the text to appear in a variable called `location`.
4) Blit the text to the screen!

***Hint:*** *Your cheat sheet will help you remember the format that these variables require!*

## Extension 2: Print the user's score!

Now that we have Flappy Bird up and running, wouldn't it be nice to tell the user how many pipes they managed to get past? Let's print the final score to the screen!

**Extension 2.1**

1) In the `for` loop where you create the list of pipes, add a new variable called `pipe['number']`, and store the value of `i`.
2) When a collision is detected, print the value of `pipe['number']` of the pipe that was collided with to the `screen`.

## Extension 3: Make Flappy Bird Fall!

In the real Flappy Bird game, the Flappy Bird is constantly falling! Update the game so your Flappy Bird does the same.

**Extension 3.1**

1) Set the initial value of the variable `moving` to `down`.
2) In the code where you check to see what arrow key was pressed, update the code so if no key is pressed the value of `moving` is `down`.

## Extension 4: Faster Flappy Bird

Some of our users are so good, they can get past all the pipes. Let’s make the game harder for them by speeding up the game the longer they play.

**Extension 4.1**

1) Create a variable called `speed` and give it a value of `1`.
2) Update the code that makes the pipes move across the screen, and subtract `speed` from `pipe[‘x’]`.
3) Create a new `if` statement where:
    a) If the user has passed less than 5 pipes, the value of `speed` is `1`.
    b) If the user has passed more than 5 pipes, but less than 10 pipes, the value of `speed` is `2`.
    c) If the user has passed more than 10 pipes, but less than 15 pipes, the value of `speed` is `3`
    d) If the user has passed more than 15 pipes, the value of `speed` is `4`.

*Is the game harder now? Play around with value of speed to make the game harder or easier!*

## Extension 5: Forwards and Backwards

Sometimes the pipes are coming at Flappy Bird at just the wrong speed! Let’s make the Flappy be able to move forwards and backwards as well.

**Extension 5.1**

1) Update the `if` statement that checks what key has been pressed.
    a) If the right arrow has been pressed, set `moving` to `forward`.
    b) If the left arrow has been pressed, set `moving` to `back`.
2) Update the `if` statement that checks what the value of `moving` is.
    a) If the value of `moving` is `forward`, increase the value of `bird_x` by `1`.
    b) If the value of `moving` is `back`, decrease the value of `bird_x` by `1`.

*Have fun sending Flappy Bird all over the screen!*

## Extension 6: Falling Obstacles

Avoid pipes is easy! Let's have some obstacles fall across the screen as well!.

**Extension 6.1**

1) Store the obstacles picture in a variable called `obs_image`.
2) Create a new `list` called `obstacles`
3) Create a `for` loop that generates the value of `obstacle['x']` and `obstacle['y']` for each obstacle you want to create.
4) Inside the game loop, create a `for` loop that blits the `obstacles` to the `screen`.
5) Update the value of `obstacle['x']` and `obstacle['y']` for each `obstacle` so that they fall diagonally across the screen
6) Detect a collision with the `obstacle`! If Flappy Bird hits an obstacle, it's game over!

***Hint:*** *If you get stuck, look back at what you did in Part 3 of the first workbook!*

## Extension 7: Collect point bonuses!

Let's give the user additional ways to collect points! Theres two different ways we can do this:

1. When "Press A now" appears on the screen, and the user presses a, we can give them bonus points; or
2. When a certain powerup, such as an apple, appears on the screen and the Flappy Bird collects it.

**Extension 7.1**

1) Create a new variable called `bonus` and set it to `False`.
2) Create a new variable called `score` and set it to `0`.
3) Create a new variable called `counter` and set it to `-1`.
4) Inside the game loop, create a new `if` statement. If the value of `bonus` is `False`, randomly choose whether the value of `bonus` is `True` or `False`.
5) Inside the game loop, create a new `if` statement. If the value of `bonus` is `true`, set a variable `counter` to `100`.
6) Inside the game loop, create a new `if` statement. If the value of `counter` is greater than `0`, `print` the words `"press a now for a points bonus to the screen"` and decrease the value of `counter` by `1`. If the value of `counter` is less than `0`, set `bonus` to `False`.
7) Inside the `if` statement you created in step 6, check to see if the `'a'` key was pressed. If it was, set the value of `counter` to `0`, increase the `score` by `5` and set `bonus` to `False`.

***Hint:*** *If you get stuck, take a look at what you did in part 2 of the first workbook!*

**Extension 7.2**

- Store the power up picture in a variable called `power_image`.
- Create a new `list` called `powers`.
- Create a `for` loop that generates the value of `power['x']` and `power['y']` for each power up that you want to create.
- Inside the game loop, create a `for` loop that blits the power ups to the `screen`.
- Update the value of `power['x']` for each power up so that they move across the screen.
- Detect a collision with the power up. If Flappy Bird collects the obstacle, add five points to the `score`!

***Hint:*** *If you get stuck, look at what you did in part 3 of the first workbook!*

## Extension 8: Add pipes to the top and the bottom!

In the real Flappy Bird game, there's pipes at the top at the bottom of the screen that you have to go through. These pipes should have a gap of at least 50 so there's space for Flappy Bird to get through!

**Extension 8.1**

- Remove the `pipe['flipped']` from the for loop where the pipes are created.
- Update the `for` loop in the game loop where the pipes are blitted. Remove the `if` statement, and subtract 50 from the `-pipe['y']` for the flipped pipe. Make sure both the normal pipe and the flipped pipe are blitted to the screen!
- Update the `for` loop in the game loop where collision with Flappy Bird are detected. Store the position of the normal pipe in `pipe_rect`, and the position of the flipped pipe in `flipped_pipe_rect`.
- Update the `if` statement where collision is detected to check if the `bird_rect` has collided with `pipe_rect` or `flipped_pipe_rect`.

You should now see pipes on the top and the bottom of the screen! The distance between the pipes may vary.

## Extension 9: Make the pipes move!

Looks like our game is still way too easy! Let's make it harder by having the pipes move up and down.

**Extension 9.1**

- In the `for` loop where the pipes are created, add a variable called `pipe['move']` and give it an empty string.
- In the `for` loop where the pipes are blitted to the screen, create a new `if` statement. If `pipe['move']` is less than 300, set `pipe['move']` to up. If `pipe['move']` is greater than 450, set `pipe['move']` to down.
- In the same `for` loop, create a new `if` statement. If `pipe['move']` is set to up, increase the value of `pipe['y']` by one. Otherwise, decrease the value of `pipe['y']` by one.

Wow! This game just got really hard!

## Extension 10: Display the High Scores!

So, our player wants to know if they've beaten the high score! Let's keep track of the 10 top high scores, and let the player know if they made the leaderboard!

**Extension 10.1**

- Create a file called `highscores.txt`. Store this file in the same folder that your program is saved in.
- In the file `highscores.txt`, on each line add a number, a comma, then a name. Save the file.
- When the game is over, open the `highscores.txt` file for reading
- Read the high scores in as a `list` called `high_scores`.
- Sort the `high_scores` to be in order.
- Print the high scores to the `screen`.
- Ask the user for their name in the shell. Make sure you tell them on the screen that you're waiting for their name! Store the user's `input` in a variable called `name`.
- Add the user's name and the current score to the list of `high_scores`.
- Overwrite `highscores.txt` to store the current list of `high_scores`.

## Extension 11: Free pass!

For every 5 pipes that our Flappy Bird gets through, let's allow them to just shoot through the next one! Make sure the player knows they have this ability!

**Extension 11.1**

- At the top of the `screen`, `print` a message telling the user that every sixth pipe won't hurt Flappy Bird!
- At the bottom of the `screen`, over each pipe, `print` what number pipe it is. Make each sixth pipe number a different colour.
- For each pipe, check to see `if pipe['number']` is a multiple of six. If it isn't, check for a collision.

***Hint:*** *Extension 1 shows you how to print to the screen.*

## Extension 12: Infinite pipes!

So, our players can constantly make it to the end of the Flappy Bird game! Let's make the game harder by always generating another 20 pipes if they make it past them all!

**Extension 12.1**

- Create a `function` called `generate_pipes`
- Move your code that creates the `list` of pipes to your new function `generate_pipes`.
- Update your game loop so that every time a user passes 15 pipes, the `generate_pipes` function is called.

## Extension 13: Game Reset!

Our players love the game so much, they just want to keep playing! Let's make it easier for them to reset the game when they lose.

**Extension 13.1**

- Create a new function called `reset_game`.
- Move all your code for setting up the game into this `function`
- Set all the variables used in the `reset_game` function to be global.
- Before the game loop, call the `reset_game` function.
- Where you check to see if the player pressed the `'q'` key, check to see if the user pressed the `'r'` key. If so, call the function `reset_game`.