**Look Before You Leap! Game tutorial**

Tasks where you need to take action are marked with a square, you can tick these off to track your progress!

If you get stuck ask for help!

Instructions marked with a ‘!’ mean it is some advanced task or knowledge, for learners who want to know or do a bit more.

1. Run the Game!

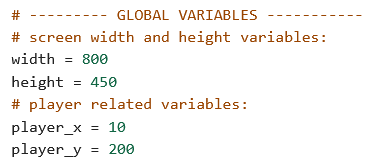
* copy and paste the provided code into your environment and give it a run! (Or have a mentor do it for you 😊), this is the game we plan on making…

1. Setting the Scene.
   * In your Code Editor you should now copy and paste the “skeleton” code, we will fill this out to produce our game!
   1. At the top of the skeleton code, in the Global Variables section, create two new variables, width = 800 and height = 450.A white background with brown text

      Description automatically generated
   * A variable is a place we can store a value in our program, this is like an organised chef who stores all their ingredients in labelled containers, and when they are cooking, they can use the food inside that container, they can also empty out that container and replace it with something else. Except in our case, the chef is our computer, and they are running a program rather than cooking: during the run time we can use the values stored in a variable by referring to their name. Rather than to store a single ingredient that likely won’t change, a programmer will use a variable to store information that is likely to be changed and edited as the program runs.
   * There are 2 types of variables, local and global. Global variables (like the ones we just created) can be seen across the whole program, while local variables can only be seen in the area of the program they were created, but we will talk more about this later…
   * In the **Setup** function, we want to create a new variable for the background colour of our game, as well as set the screen size we want our game to fill: A number and text on a white background

     Description automatically generated
     + A function is just a name we give to a block of code that performs an action in our program, they are useful if we want to use it multiple times or make the purpose of our code clearer. Here the function is defined for us, but we will make our own later…
     + RGB values explanation…
     + We now need to run the code we have just written. The run() function will run setup followed by draw and it will continue to run draw until the program is stopped by the user.
     + When we use a function in Python, we often use the term “call”, so an experienced programmer might say to call the run function at the end of the program.
     + Add the run function at the end of the program with the frame\_rate set to 15: A black and green text

       Description automatically generated with medium confidence
     + Your code is now ready to run! Press the Run button at the bottom of the screen: A white rectangular object with a blue border

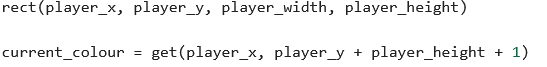
       Description automatically generated
     + In Python you can write comments to yourself by starting a line with a ‘#’, have a look and see how they’ve been used previously in the skeleton code, then try and come up with some comments of your own for everything we’ve done up till this point!
     + What do you think the effect would be in you changed some of the numbers in the Color function? Have a go and find out, we will use this knowledege later…
2. Drawing our player!
   1. In the global variables area of our code in the player related values section create two new variables player\_x and player\_y: 
   2. Define a new function draw\_player below the draw function. A close-up of a number

      Description automatically generated
   * We can’t use spaces when defining the names of functions or variables so common practise in Python is to use \_’s (aka underscores) but not all languages follow this! In fact some languages encourage using a capital letter for each new word, so in our case we would call our function drawPlayer()!
   * We have defined a few global variables so far, this means within any of our functions we can use these variables. BUT we cannot change them within these variables as the changes would not be permanent, to make permanent changes to these global variables we need another line of code.
   * Add global variables player\_x and player\_y to the draw player function: A close-up of a text

     Description automatically generated
   * Below this, still within the draw\_player function, set the player colour and dimensions and draw the player on the screen: A screenshot of a computer code

     Description automatically generated
   * This code draws a uses the values (255, 0, 0) to draw a red player, these values represent RGB or Red, Green, Blue, this tells the computer the amount of these colours to use, try to change these values until you get a colour you are happy with!
   * Finally, we want to add draw\_player to the draw function, so the code we have just written gets used when we run our program. Try to figure out how to do this by looking at how we use draw\_level in the draw function.

1. Making our player fall??!
   * Now we are drawing a player to our screen we want to make them move, to do this we are going to write some code so that if there is no platform underneath our player that they fall down the screen until they collide with a platform.
   * Before we begin how do you think we could try to figure out if there is a platform underneath a player? Hint: it may involve using the pixels in our level.
   1. In the global player related variables create new function called falling and set it to True.
   2. Now in the setup() function, we want to turn bgd\_colour into a global variable so we can use it in the draw player function: A number and text on a white background

      Description automatically generated
   * It’s common practice to define global variables at the beginning of your code, and if defined within a function to define them at the beginning of the function.
   * In draw\_player we want to get the current colour below our directly below our player, to do this we will use the ‘get’ function which will return the colour for a given (x,y) coordinate: 
   * Next, we need to write an If statement to check if the colour of the pixel below our character is the equal to the background colour: A black text on a white background

     Description automatically generated
   * Here we compare the hex codes of the colours, hex stands for hexidecimal and is a way to write big numbers with a small number of characters, we use them here because hex gives us a string representation of the colour which is easy to compare to other strings.
   * If that condition is True, we want to run some code to lower the position of our character, and indicate that they are currently falling: A white background with black and red text

     Description automatically generated
   * Now we want to add some code to execute if there IS a platform beneath us, to do this we could use another ‘if’ statement, but instead we are going to use something called an ‘else’. This is a bit of code that executes when our ‘if’ doesn’t. It may be helpful to think of it as “**if** something is true **then** do some action **else** do some other action”.
   * Add an else to your if statement that indicates the player is no longer falling: A white background with colorful text

     Description automatically generated
   * Next, we want to add some code that executes when our player falls in lava, to do this we will use an **else if,** in python we write this as “**elif”**. This is some code that will run after the first if, and before the else. We use it so that we can write another condition to execute some code, rather than an else which will always run. It can be helpful to think of the flow of a full if statement including an if, elif and else as “**if** something is true **then** do some action **else if** another thing is true **then** do some other action **else** neither of these were true, do another action.”
   * Explain the += function
   * Add an elif to print a message if the player falls into lava, feel free to replace game over with whatever message you want: A computer code with colorful text

     Description automatically generated
   * Now run the program and test if it works – we should see the player fall and land on the starting platform.
2. Game Over
   1. Define a new global player-related variable called game\_over and set it to True.
   2. Add game\_over to the global variables in draw\_player(), remember we need to do this if we plan on changing the value of the game\_over variable across the whole program: A black text on a white background

      Description automatically generated
   3. In the elif of our draw\_player if statement, set game\_over to True, we will use this to indicate in another area of the program that the game has ended. A close-up of a computer code

      Description automatically generated
   4. In the draw() function, create a new if statement to only do draw\_player() if game\_over is equal to False. (You don’t need an else or elif)
   * Test your program by setting player\_x in the global variables to 60 and running your code, before you run what do you think will happen? Is what does happen what you expected? (remember to set it back to 10 when you are done!)
3. Making our player jump!
   * One of the most important skills in programming is learning how to use other people’s code. Why spend hours finding the best way the to write some difficult code when someone out there has already done it and made it publicly available? Here we are going to use try to understand the purpose of some code and use it to make our player jump.
   1. First create 4 new jumping related global variables, setting them to default values or taking an input from the user: A screenshot of a computer

      Description automatically generated
   2. Explain why we use int here
   3. Explain why we use -1 on the jumps\_left input
   4. Add all these variables to the global variables in draw\_player: 
   5. Explanation of the not operator…
   6. Explanation of the and operator….
   7. Next add another if statement to indicate the player should jump if they are not falling and they are not jumping: 
   8. Now we need to edit some of our previous code to fit in with the jumping mechanism we are about to implement. Currently our if statement that makes the player fall has no knowledge of if the player is jumping or not, this means if the player jumps, they will fall mid jump! How do you think we can modify the line below to also check if the player isn’t currently jumping? (hint: look at how we do this in the jumping if statement)
   9. Add an elif to our jumping if statement to check if the player has initiated a jump: A close up of a math equation

      Description automatically generated with medium confidence
   * Now, take a look at the jump function in the skeleton code. We can see that it takes one “argument” called jump\_progress, this is used in when jumping to make the jump motion happen in steps rather than all at once. The reason for this is so that we have an animation for our jump, as without this step our jump function would just teleport us to the end of the jump, and our game would look kind of weird.
   * Relate this the game design
   1. Inside this elif we want to add another if statement to check if jump progress is less than 10, this indicates we are mid-jump. Then if we are mid jump, we want to call the jump function and increment jump\_progress: A close-up of a math equation

      Description automatically generated
   * Explain the term increment?
   * Explain the term nested if
   * We now have code to handle when makes our player start jumping, but what about when we finish a jump?
   1. Add an else to the jump progress statement to stop jumping once we finish a jump, to set the jump progress back to 0 and reduce the jumps left by 1: A screenshot of a computer program

      Description automatically generated
   * We can see now with these nested if statements our code is starting to get complicated! Keeping track of whether something is within another if statement gets tricky when using nested if statements. In the code we’ve just written, what change can we make to this code to get rid of one of our nested ifs? Hint: it requires using an elif. A white background with black and green text

     Description automatically generated
   * You can now try out you code and add some values to control the jumping of the player, but currently if we do reach the finish nothing will happen! Next we will add some code to make our game winnable…
4. Letting our player WIN!
   * Briefly explain returning from a function.
   * Now look at the check\_if\_finished function from the skeleton code. This function uses a similar method to our platform detecting code, but this time it detects if we touch the green flag. The function “returns” True if we do touch the green flag and False otherwise. We are now going to use this function to make our game winnable…
   1. In the draw function, add finished to the global variables: 
   2. Next create an if statement to check if the game not finished, and put the code for drawing the level and the player inside it: A computer code with text

      Description automatically generated
   3. Finally, create an else statement to run some code when the game does finish, this code adds some text to the centre of the screen, feel free to change the message to whatever you want! A close-up of a computer code

      Description automatically generated
   * If you played around with the jumping earlier or have being paying close attention to the code you will notice that when we finish jumping our player just stops moving, this makes it hard to complete our level as we would need to find the exact position on the flag to win the game. Now we are going to make the game a bit easier by adding a walking function to our player, so when the reach the final platform they can just walk to the finish line.
   1. At the end of the draw\_player function add an if statement t check if the jumps left are less than or equal to 0, to stop jumping and to start walking: A computer screen shot of a code

      Description automatically generated
   * We should now have a working game! Give it a try!
5. Tuning and tweaking!
   * Now that you have finished programming the game, try to complete it!
   * Can you figure out where in the code you would edit to change the layout of the platforms?
   * Have a go changing the framerate of our game, what do you think the effect will be? What is the actual effect? Think about why this is useful to have control over.
   * Try changing the layout of the game by changing the variables related to the platform. Can you still complete it?
   * Look back at the global variables we defined, what would happen if we changed some of them? Have a go and see if you were right!
6. QUIZ!!!
   1. What would the following if statement print? A screen shot of a computer code

      Description automatically generated
      1. A
      2. B
      3. C
      4. B C
   2. What would the following code print? A computer screen with text

      Description automatically generated
      1. I LOVE FUNCTIONS, HEY GUESS WHAT?, I LOVE FUNCTIONS, Oh yea I do….
      2. HEY GUESS WHAT?, Oh yea I do….
      3. HEY GUESS WHAT?, I LOVE FUNCTIONS, OR DO I, Oh yea I do….
      4. HEY GUESS WHAT?, I LOVE FUNCTIONS, Oh yea I do…
   3. What the following code print? A screenshot of a computer

      Description automatically generated
      1. 20
      2. 25
      3. 55
      4. 60
   4. What would the following code print? A computer screen shot of text

      Description automatically generated
      1. hello!
      2. goodbye!
      3. guten tag!
      4. Nothing….