Level 2 - Python Projects

1. Rock Paper Scissors with Computer

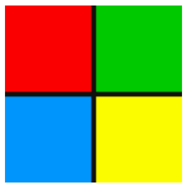
Create a program that simulates the popular hand game: Rock, Paper, Scissors, usually played between two people, in which each player simultaneously forms one of three shapes with their hand.

Notes:

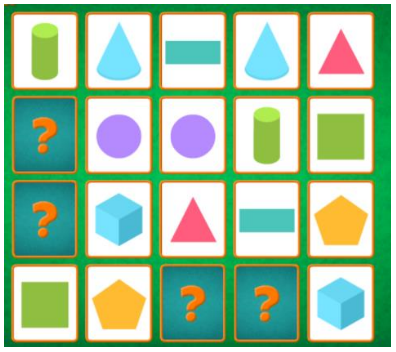
- Your Game should be between **1 player** and the computer
- The game begins with a user (player) getting to choose a hand form **by typing it in** or clicking on an image.
- The computer will immediately output: the users choice, the computer's choice and report who won that round.
- The computer's choice should be selected **randomly** from a **list**
- The game should be able to be **repeated until the user decides to quit** by entering a particular word or key stroke.

2. Simon Says Game

- Create a game that will display four colored squares on the screen.
- The game will then flash/illuminate a series of colors in a particular order and **ask the user to copy the order.**
- The player can either **type in the order of colors** they saw or **click on the squares** in the correct order to indicate the pattern they saw.
- If the player gets the pattern correct, the computer will flash another pattern for the user to follow.
- After a few successful attempts by the user the patterns can either become faster, longer or both.
- The game should proceed so the complexity/speed of the patterns increases in "levels".
- The player get more points (or just bragging rights) depending on the level that they can achieve.



3. Memory puzzle



- This should be a **two** player game.
- When the **game** starts, a series of **tiles** are displayed face down on the screen.
- The first player then flips over two tiles, selecting them by clicking on them.
- If the two **tiles** have the same image, the cards are removed and you get a point. If they are not the same a **second** player takes a turn.
- The game is finished when all the matches have been made
- The game should keep track of how many matches each player has made and report the winner to the players

4. Hang man Classic

Create the classic hangman game where a series of underlined spaces are displayed that represent the unknown letters of a mystery word.

Notes:

- The user guesses a letter. If that letter is in the mystery word the letter will appear where it should above the blank underline in the word. If the guessed letter does not appear in the mystery word, a portion of the hang man drawn and the incorrect letter is reported at the side of the game.
- This continues until the player has all the letters in the word or they have made the maximum amount of guess necessary to end the game in defeat.

5. Tick Tack Toe

Two players: (X and O), **take turns** marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner. **This should be a two player game.**

6. Speed Typing Game

A paragraph appears on the screen. The user has to retype the paragraph as quickly as possible without making a mistake. You may choose to award points one of two ways: based on the **time it takes** to type the paragraph **or** if you are able to complete the paragraph in a **predetermined** span of time.
The game should include multiple chances (or levels) for the user. Hint: you will have to import the time library for this game.

7. Video game

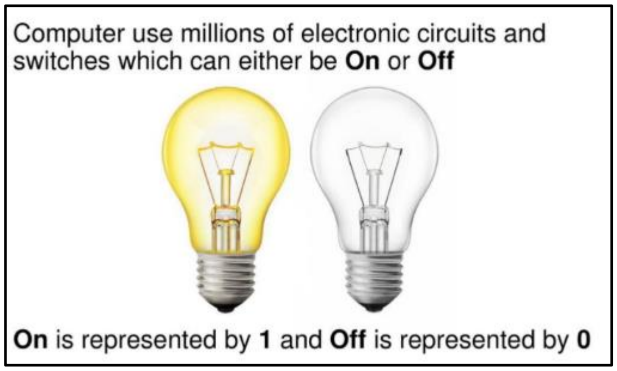
Review the mini video game shown to you in the Leve1 – Python Turtle assignment.
Using that lesson and other resources you can find, try to create any video game you wish.

8. Black Jack

Black Jack is one of the simplest casino card games and a great project for computer programming students. Look up how the game is played (play a few rounds yourself online if you wish). Then use the model below to create a black jack game that your friends can play.

1. Create a deck of 52 cards
2. Shuffle the deck
3. Deal two cards to the Dealer and two cards to the Player
4. Show only one of the Dealer's cards, the other remains hidden
5. Show both of the Player's cards
6. Ask the Player if they wish to Hit, and take another card
7. If the Player's hand **do not** go over 21, ask if they'd like to Hit again.
8. If a Player Stands, play the Dealer's hand. The dealer will always Hit until the Dealer's value meets or exceeds 17
9. Ask the Player if they'd like to play again.

9. Binary to Decimal and vice versa

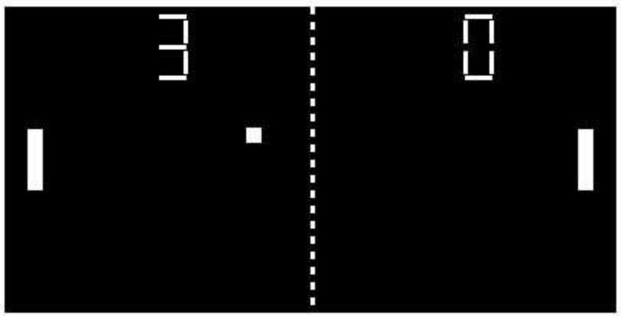

Computer use millions of electronic circuits and switches which can either be **On** or **Off**
On is represented by **1** and **Off** is represented by **0**

Computers operate by manipulating on and off electrical signals. A number system called **binary** is used to encode the on and off (0 and 1's) into data that we can store and use.

Look up how the **Binary System** works using the links on the course page and learn how to **convert** Binary to Decimal and the reverse.

Create a program in python that can convert binary numbers to decimal numbers (and the reverse).

10. Pong

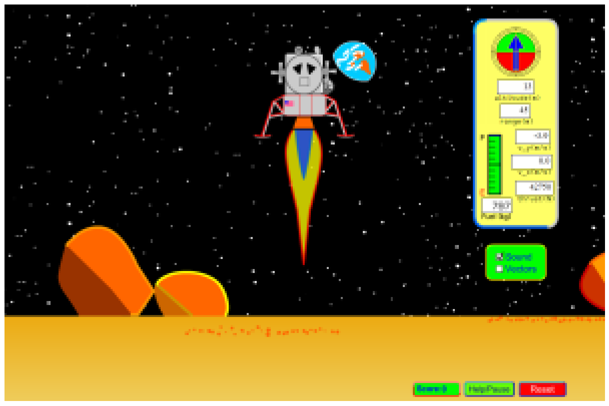


Create the classic game pong! Not familiar with this arcade classic? Check out: https://pong-2.com/

Lots of options here. You can try:

- 1 or 2 player versions.
- A version that has different levels with different ball speeds.
- A version that makes ball's direction (as bouncing off the paddle) depend on the direction the paddle is going.

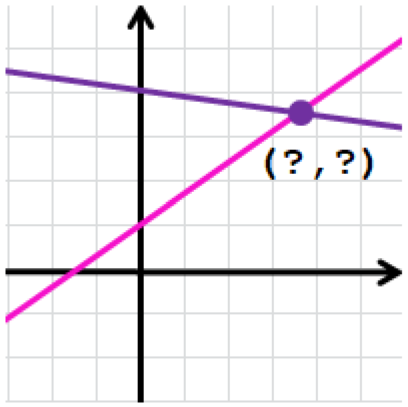
11. Lunar Lander



Create the classic game Lunar lander!
Not familiar with this arcade classic?
Find your own version online or checkout:
https://scratch.mit.edu/projects/72798680/

Lots of option here. Best to play a few online versions so you can see all the cool stuff you can add.

12. Intersection Point



Write a program that will allow the use to input the equations of **two** straight lines:

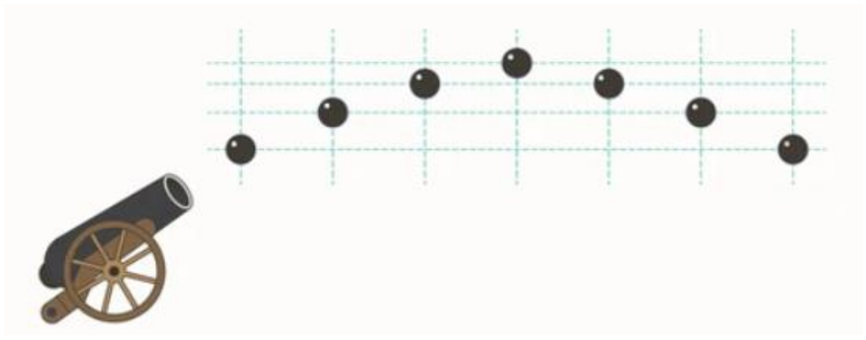
Line 1: y=*a*x+*b*
Line 2: y=*c*x+*d*

First the user will choose the slope and y intercepts of each line (a,b,c,d) above.

Then the program will then calculate the coordinates of the intersection point of the tow lines (if such a point exists)!

13. Projectile Motion Simulator

Create a projectile motion simulator (*without referring to a python tutorial*).



Your simulator should let the user select an **initial *speed*** and ***angle*** and then show an accurate path and *motion* the projectile will take given those inputs.

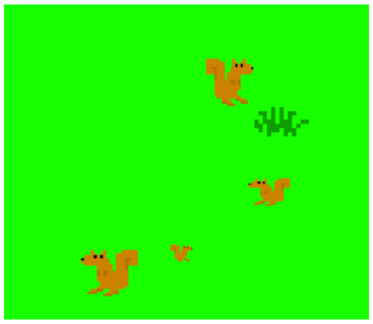
Example simulator:
https://phet.colorado.edu/sims/html/projectile-motion/latest/projectile-motion_en.html

You can make your simulator into a **game** or make any additions you wish as long as the motion the projectile is accurately modeled by the correct physics.

If you have not taken Physics 11 or are not familiar with the physics of projectile motion, then talk to Mr. Walzl. Or you can try to get an idea for the physics using any online resources you wish (some OK ones below):
https://www.youtube.com/watch?v=aY8z2qO44WA

https://www.physicsclassroom.com/class/vectors/Lesson-2/Horizontal-and-Vertical-Components-of-Velocity

14. Squirrel Eat Squirrel



Create the following game. The player controls a small squirrel that must hop around the screen eating smaller squirrels and avoiding larger squirrels. Each time the player's squirrel eats a squirrel that is smaller than it, it grows larger. If the player's squirrel gets hit by a larger squirrel larger than it, it loses a life point. The player wins when the squirrel becomes a monstrously large squirrel called the Omega Squirrel. The player loses if their squirrel gets hit three times.

15. **Decoding a Message (find the Walzl-dollars):**

Mr. Walzl has decided to hide a stack of Walzl-dollars at an unknown location.
He has left a sequence of coded instructions for his assistant so that the assistant can retrieve them. Each instruction is a sequence of **five digits** which represents a direction to turn and *the number of steps to take*:

The first two digits represent the direction to turn:

• If their sum is odd, then the direction to turn is left.
• If their sum is even and not zero, then the direction to turn is right.
• If their sum is zero, then the direction to turn is the same as the previous instruction.

The remaining three digits represent the number of steps to take which will always be at least 100. Your job is to create a program for the assistant so they can decode the instructions and always find the secret formula.

Input Specification
There will be at least two lines of input. Each line (except the last line) will contain exactly five digits representing an instruction. The first line cannot begin with 00. The last line will contain 99999 and no other line will contain 99999.

Output Specification
There must be one line of output for each line of input except the last line of input. These output lines correspond to the input lines (in order). Each output line gives the decoding of the corresponding instruction: either right or left, followed by a single space, followed by the number of steps to be taken in that direction.

Sample Input
```
57234
00907
34100
99999
```
Output for Sample Input
```
right 234
right 907
left 100
```