Level 3 – Senior Projects

1. Battleship Game

Create a battleship using turtle graphics (or other graphics tool). The idea is to build a square matrix the program will secretly place battleships randomly on the grid. The user has a few chances to drop a bomb on the location of the ship by guessing. After a limited number of guesses, the game will be over. You may also wish to create the traditional two player battle ship game.
If you are not familiar with battleship try the game out at: https://www.battleshiponline.org/
Some sample code @
https://www.101computing.net/battleship-challenge/

2. Binary Search Algorithm

Binary Search is a search algorithm that can be used **find** a specific element in a list (array) and its position **quicker** than just comparing **every** element in the list with the specific element you are looking for. One important condition for using the binary search is it can **only be used on a sorted list.**

Example:

Let's say we wanted to find 84 in the **sorted** list below. The most obvious way would be to start with the first element (on the left) in the list and compare it to see if it is 84, then repeat the process for each element. When the `list[x]`==84 we know we have found 84 and its position.

6	12	17	23	38	45	77	84	90

Sounds great, but it is really **slow**! And when we are talking about billions of data elements **speed matters**!

There are many search algorithms are designed to **reduce the time needed** to do specific tasks. Even though computers are incredibly fast now a days, if you aren't careful how you do things, your program may take days to complete its desired mission.

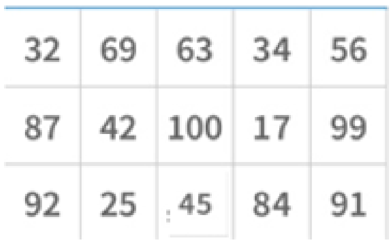
Use the links on the course page to learn how a **binary search** works and then create two programs:

1. Create a binary search that can find the position of a integer in a sorted list of 30 numbers
2. Create a binary search that can find a word in a alphabetically sorted list of 30 words

3. Covid Concert

During Covid outdoor music concerts have become an orderly affair where people have to stand spaced out in a perfect 2-D grid with a number of rows and columns.

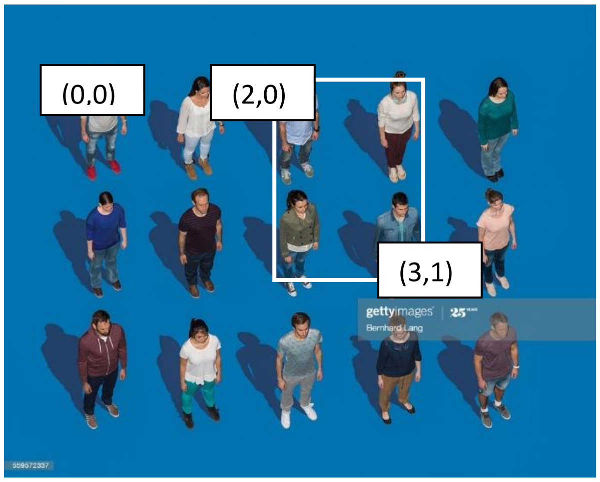
Post Malone decides at some of his concerts to eject **older** people in different sections of his concern (*because they ain't cool*).



32	69	63	34	56
87	42	100	17	99
92	25	45	84	91

Create a program that can generate a 2-D list (or array) that is populated with the ages of concertgoers (ages 16 to 100). Ages of the concertgoers should be generated randomly. Have the user chose the size of the 2-D array (max 100 x100). *Example input*: 4 x 3

The user can now be prompted to select *any* **sub section** of the array.

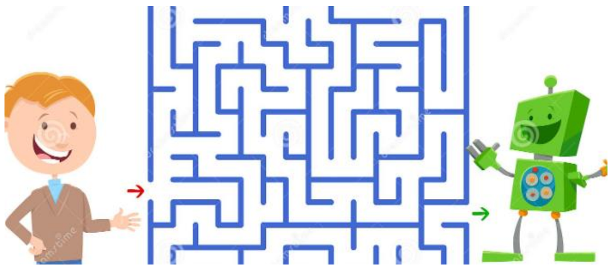


Example input:
If locations start at top left hand corner of full array (0,0):
top left hand corner of sub array: (2, 0)
bottom right hand corner of array: (3,1)

The program will now *find* the oldest person in the selected *sub-section* so they can be ejected.

Bonus if you can have an accurate visual/animated representation of the scenario.

4. Hungry Robots Game



Create a maze game with hungry robots! The robots are badly programmed and **will move directly toward you, even if blocked by walls**. You must trick the robots into crashing into each other (or dead robots) without being caught. You have a personal teleporter device that can send you to a random new place, but it only has enough battery for two trips.

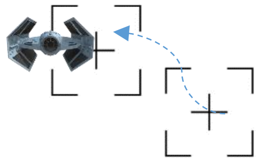
Hints and Details:

a) First start by creating a **simple** maze that a user can navigate through use the arrow keys on the keyboard.
b) The character should start a one point in the maze and will win if they can escape through one of several exits.
c) The maze hallways should be wide enough for your character to maneuver around the robots if your are quick enough.
d) Robots that touch another live or dead robot will also become dead and will be left in the place where they die. A change in color should indicate if a Robot is dead.
e) Your player CAN touch dead robots without consequence, but cannot move through them. If you touch a live robot you are dead.
f) Program the robots to always move directly towards you even if there is a wall between you and the robots. Robots **cannot** move through walls.
g) Start the game with robots randomly distributed about the maze
h) You have a personal teleporter device that can send you to a random new place, but it only has enough battery for two trips.
i) **Additional *optional* features**: Give the player a limited number of traps they can leave behind to stop any robot that steps on one. Give the player a limited number of "instant walls" that they can put up for their own defense

5. Connect 4

Create a python text game or pygame that simulates the popular connect 4 game. See this link for an example
https://www.cbc.ca/kids/games/all/connect-4

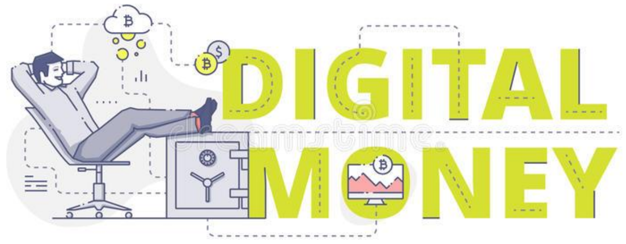
6. Target Detection



Create a computer program that can detect if objects that fly across the screen, are in the sight of the user's weapon.

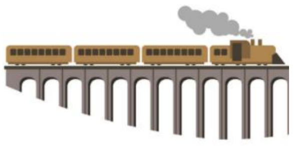
a) The user's sight is controlled by the user and can move around the screen with the *mouse* or arrow keys
b) A detection should signaled to the user with a change of color of the sight and written indicator "ON TARGET FIRE!"
c) As enemy objects fly around the screen, they should have a random element to their movement.
d) "detection" should only be signaled *if every pixel of the enemy lies within the entire weapon sight.*

7. Digital Walzl-dollars



Using Python create a system that can generate and keep track of digital Walzl-dollars. Your concept must be secure and easy to use. **Example:** generate unique QR codes. When scanned by a laptop camera the program should identify who is the rightful owns the QR code. Security features should be paramount and *considered carefully*. You may use any system/idea you wish if you can prove it is secure, effective, and easy to use.

8. Train Bridge Problem:



Create a program that will tell you the maximum number of railway cars that can be brought across a rail way bridge before is will collapse.

About the Bridge:

- The length of all train cars is 10m. The bridge is 40m long (thus the bridge can only hold a maximum of 4 train cars at one time).
- The bridge will crack if the total weight of the cars on it at one time is *greater* than **100 tons**.

About the Trains:

- Each train that crosses the bridge is different.
- The weights of each **train car** in each train is different (depending on what's in them)
- For every train that crosses, the cars should be numbered starting at 1, going up to the total number of cars, and they must cross the bridge **in that order** (i.e., 1 immediately followed by 2, which is immediately followed by 3, and so on).

When a user inputs the train car weights in the correct order, it will be your job to determine how many train cars can travel safely without the bridge collapsing.

Input Specification:

The first line of input is the number N (1 ≤ N ≤ 100) which is **the number of railway cars** included in the particular train that we wish to move across the bridge.
On the next line of input the user will **list the weights of each train car** *in order* with spaces in between each weight. *Weight of individual cars must be between 10 and 60 tons.*

Output Specification:

Your output should be the maximum number of railway cars that can be brought successfully across the bridge to the other side before the bridge collapses.

Sample Input#1
6
50 30 10 10 40 50
Output for Sample Input#1
2

Explanation of Output for Sample Input#1
The first four railway cars have total weight 50 + 30 + 10 + 10 = 100, which is not greater than what the bridge can hold**. The first railway car can leave safely**. The next railway car comes on the bridge, we have a total weight of 30 + 10 + 10 + 40 = 90, which is not greater than what the bridge can hold. **The second railway car can leave safely.** The next car enters the bridge and we have 10 + 10 + 40 + 50 = 110 which is greater than the bridge **can hold. So, only the first 2 railway cars can be taken across the bridge safely to the other sided.**

Sample Input#2
8
30 60 30 10 10 20 10 20
Output for Sample Input#2
0
Explanation of Output for Sample Input#2 When the first 4 railway car enters the bridge, its weight of the train is 30+60+10+10=130 tons which will exceed the maximum weight the bridge can hold. Thus, we cannot bring any railway cars safely across the bridge to the other side.

9. Online College Admission Management System

(Perhaps a good way to practice ***object oriented programming***)

Create a college admission management system that is designed to:

1. Make the admission process much easier for students
2. Maintain a database of information in an efficient way.

Administrator side:

In this system (using the correct username and password) a college **administrator** can add all the college and program details they wish.

Each submission must include *at least* the following:

- Name of college
- Name of program
- Size of program
- Previous year's cut off mark for admissions.
- This year's cut off **(Not visible to students)**

Student side:

In the system, students can sign up by **creating** a secure user name a password and can apply for up to 3 collages or programs.

When applying for a particular program, they must submit:

- Name:
- Date of birth:
- Where they live:
- Final overall high school grade average out of 100
- Answer a few multiple choice questions about extra-curricular activities, volunteer experience, and leadership.

Result of Student Applications:

The system will inform each applicant of their success based the following:

- The system will make a list of all the applicants grades in order (highest to lowest) and reject the *bottom third* of the all applicants without further consideration (regardless of program cut-offs).
- The system will reject student who did not make the mark cut off for the program/s they selected.
- The system will reject applicants based upon some of their answers to the multiple choice questions (example: no extra-curricular activities, no volunteer experience, and no leadership could mean no acceptance)
- It will issue acceptance letters to those students who meet the mark cut-off of the program/s they are applying for **and** meet any criteria created by the multiple choice questions (specific to that program).

Scholarships:

- It will award scholarships to students in the top 5 percentile of grades and who answer the multiple choice questions "correctly".

Gaining access to application results:

- Once the student submits their application the system will tell the student that they can check back in two weeks (but for the sake of this exercise students applications should be processed immediately upon creation. When they logout the student must login **two** more times to see their results. The **first time** they login **after** they've applied, the system will tell them that their application hasn't been processed yet.
- The result of their application should presented to the user in a few sentences that either welcomes them to the school/program or kindly tells them they were rejected.