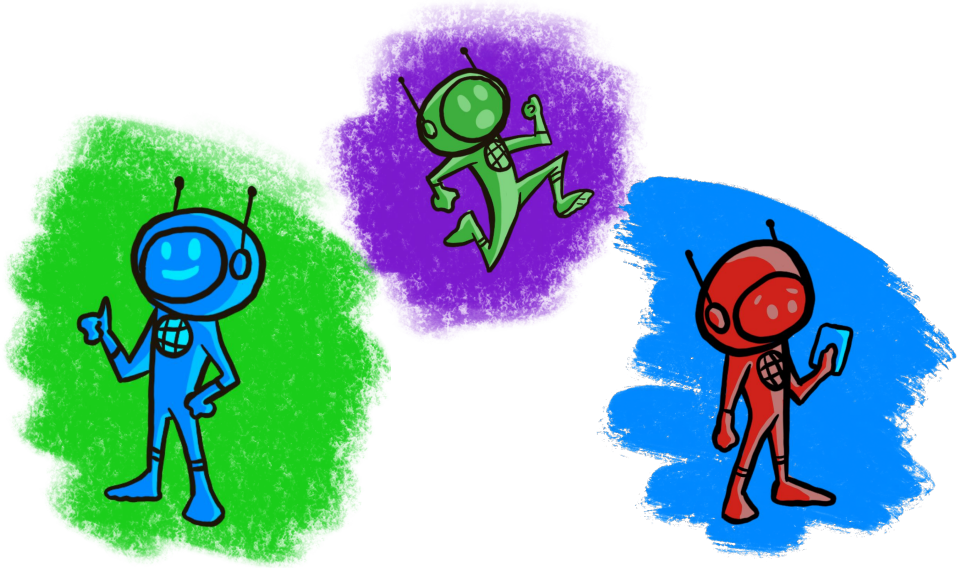
Who Goes There?

- You have been tasked with implementing *"Who Goes There?"* a program that will simulate a perilous journey through space on a spaceship.
  - The ship is crewed by a group of crewmates.
  - Each crewmate is assigned a series of tasks to complete on the journey to keep the ship running.
  - Unfortunately, the crew has been infiltrated by one or more *imposters* that will *murder* the crew members if they catch them alone.
- The simulation ends when the surviving crewmates finish their tasks *or* the imposters wipe them all out.



Does this all sound *familiar*? It should! It is the plot of classic Sci-Fi movies like *Alien* and *John Carpenter's The Thing*.

```
class Task:

    __slots__ = [“__name”, “__location”]

    def __init__(self, name, location):

        self.__name = name
        self.__location = location

    def __repr__(self):

        return(f"{self.__name} in {self.__location}")
```

Problem Solving 1

The brave crew in *Who Goes There?* must complete ***tasks*** on their journey to keep the ship in running condition.

A *task* has a descriptive ***name***, e.g. `"Fix Wiring"` and a ***location*** on the ship, e.g. `"Electrical"`.

Write a class to represent a ***task***. Follow best practices including:

- Slots
- A constructor
- Proper encapsulation
- a compact string representation in the format `"<name> in <location"`, for example `"Fix Wiring in Electrical"`

Problem Solving 2

A ***crewmate*** has a spacesuit with a unique ***color*** and a collection of ***tasks*** that the crewmate needs to complete.

The tasks are assigned to each crewmate in ***reverse priority order***, meaning that the lowest priority tasks are assigned ***first***. The crewmate should complete the tasks in ***priority order***.

Write a class to represent a ***crewmate***. Follow best practices including:

- Slots
- A constructor
- Proper encapsulation
- A compact string representation in the format `"<color> Crewmate"`.
- A method to assign a task to the crewmate.
- A method to get the crewmate's next task.

```
class Crewmate:

    __slots__ = [“__color”, “__tasks']

    def __init__(self, color):

        self.__color = color
        self.__tasks = []

    def __repr__(self):

        return(f"{self._color} Crewmate")

    def assign_task(self, task):

        self.__tasks.append(task)

    def get_task(self):

         return(self.__tasks.pop(0))
```

```
class Ship:

    __slots__ = [“__tasks”, “__locations”]

    def __init__(self, tasks):

        self.__tasks = tasks
        self.__locations = {}
```

Problem Solving 3

A ***spaceship*** has a specific collection of ***tasks*** that must be completed every journey. You should remember that every task includes a name and a location. The ***locations*** on the ship can be derived from the tasks. Keep in mind that there may be more than one task to do in each location!

Begin writing a class to represent a ship. Follow best practices including:

- Slots
- A constructor
- Proper encapsulation

What kind of data structure will you use to store the ship's locations?

Problem Solving 4

The crew quickly determines that they are being stalked by one or more imposters and decides to group together in the ship's ***cafeteria***.

They decide to venture into the ship, *one at a time* and attempt to complete tasks. If a crewmate is successful, they return to the cafeteria to wait for their next turn.

Assume that you have a list of crewmates that have already been assigned tasks. Write the code that does the following:

- Create a data structure to represent the ***cafeteria***.
- Add all of the crewmates to the data structure.
- In a loop:
  - Get the next crewmate.
  - Print the crewmate's next task.
  - Send them back to the cafeteria.

How do you determine whether or not the crewmate has any remaining tasks?

```
def main():

      Cafeteria = []
      Cafeteria.append(crewmates)
      while True:
            crewmate = Cafeteria.pop()
            task = print(crewmate)
            Cafeteria.append(crewmate)

main()
```