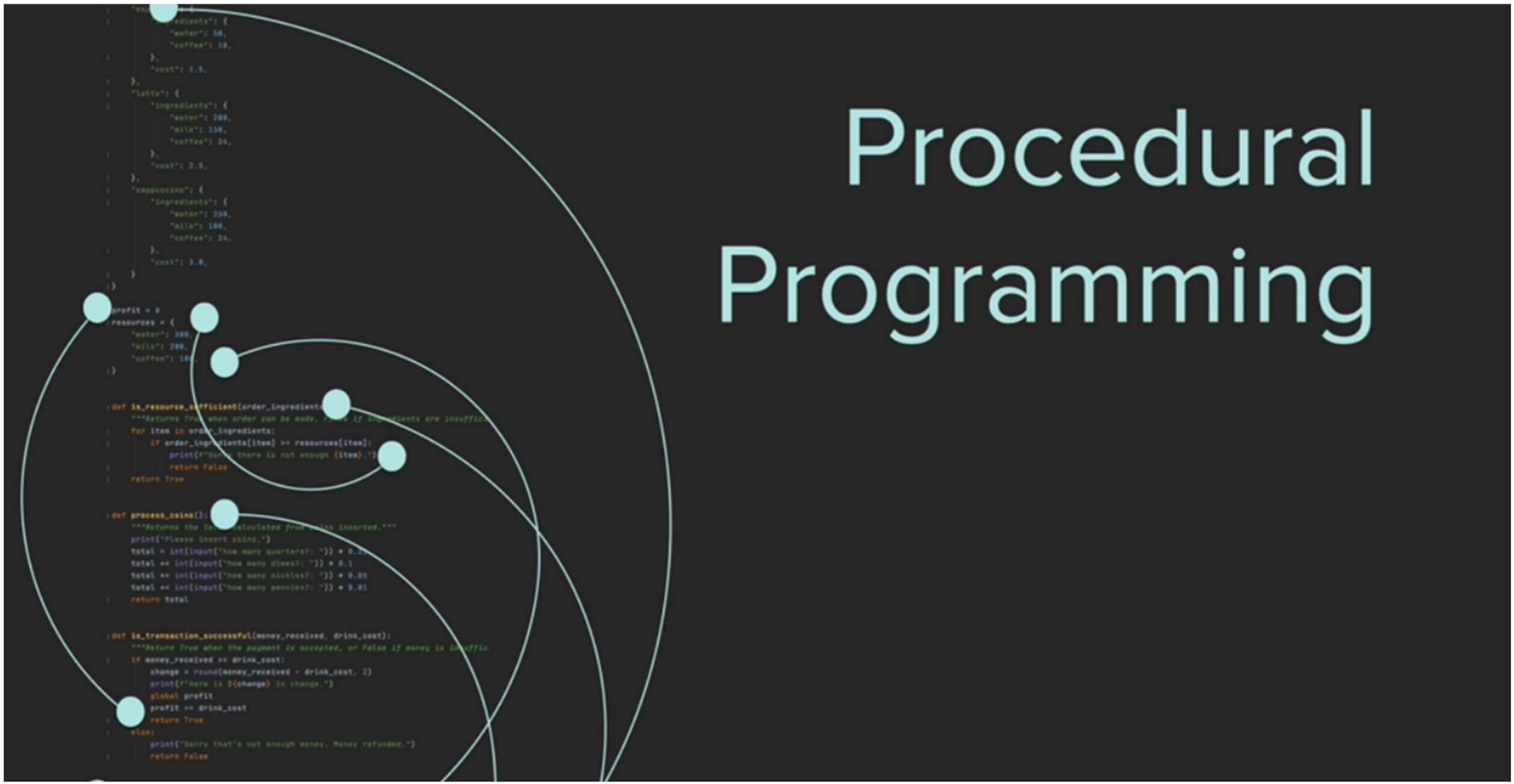
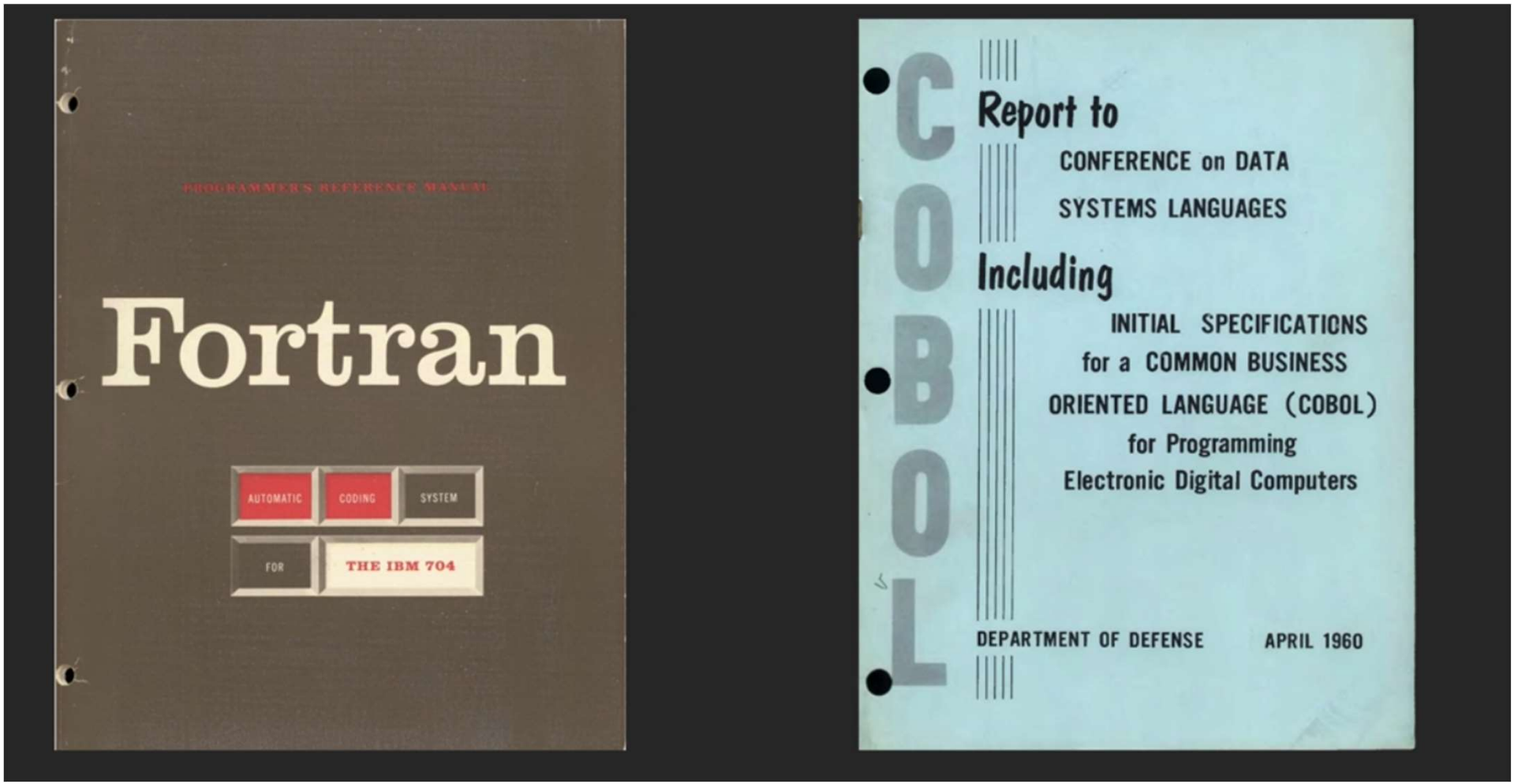
# Programming for AI

Abdul Haseeb

BS(AI)-IV

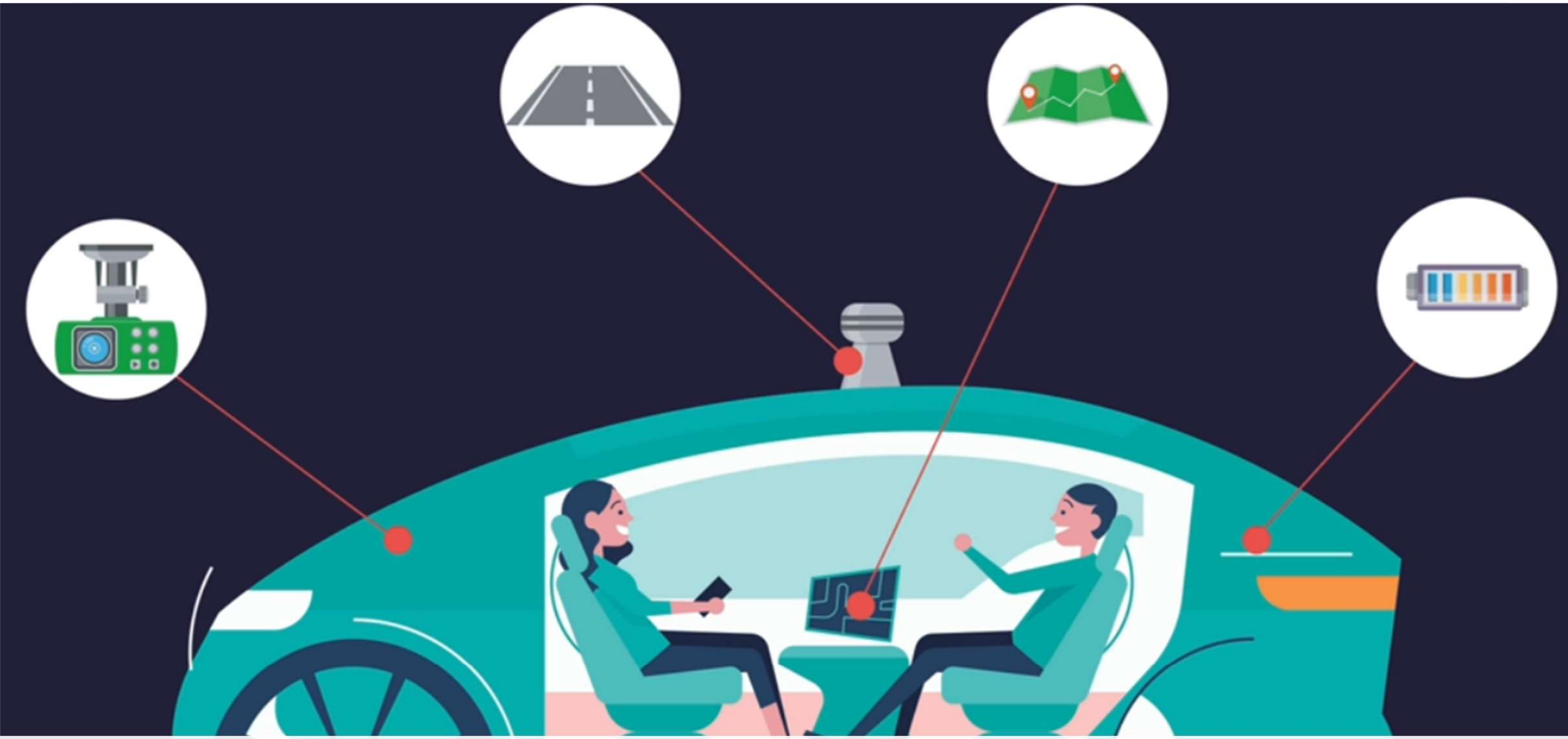

Procedural
Programming


PROGRAMMER'S REFERENCE MANUAL
Fortran
AUTOMATIC
CODING
SYSTEM
FOR
THE IBM 704
COBOL
Report to
CONFERENCE on DATA
SYSTEMS LANGUAGES
Including
INITIAL SPECIFICATIONS
for a COMMON BUSINESS
ORIENTED LANGUAGE (COBOL)
for Programming
Electronic Digital Computers
DEPARTMENT OF DEFENSE
APRIL 1960

# This is where OOP Comes handy…

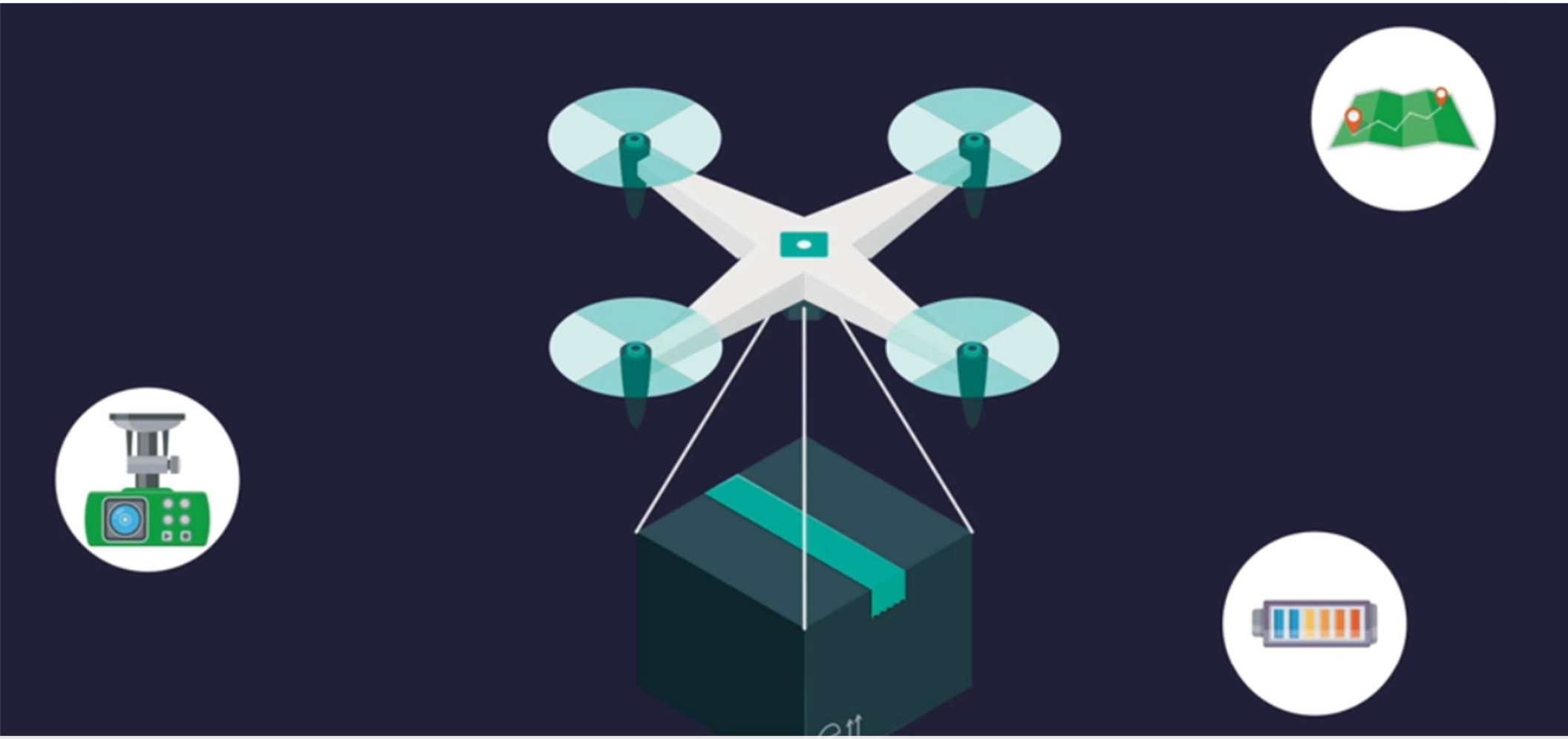
- We have to keep programming simple even writing huge amount of code.


Imagine: You have been assigned program for
self driving car.



# Breaking down into simpler problems

# Increase Productivity by working on modules

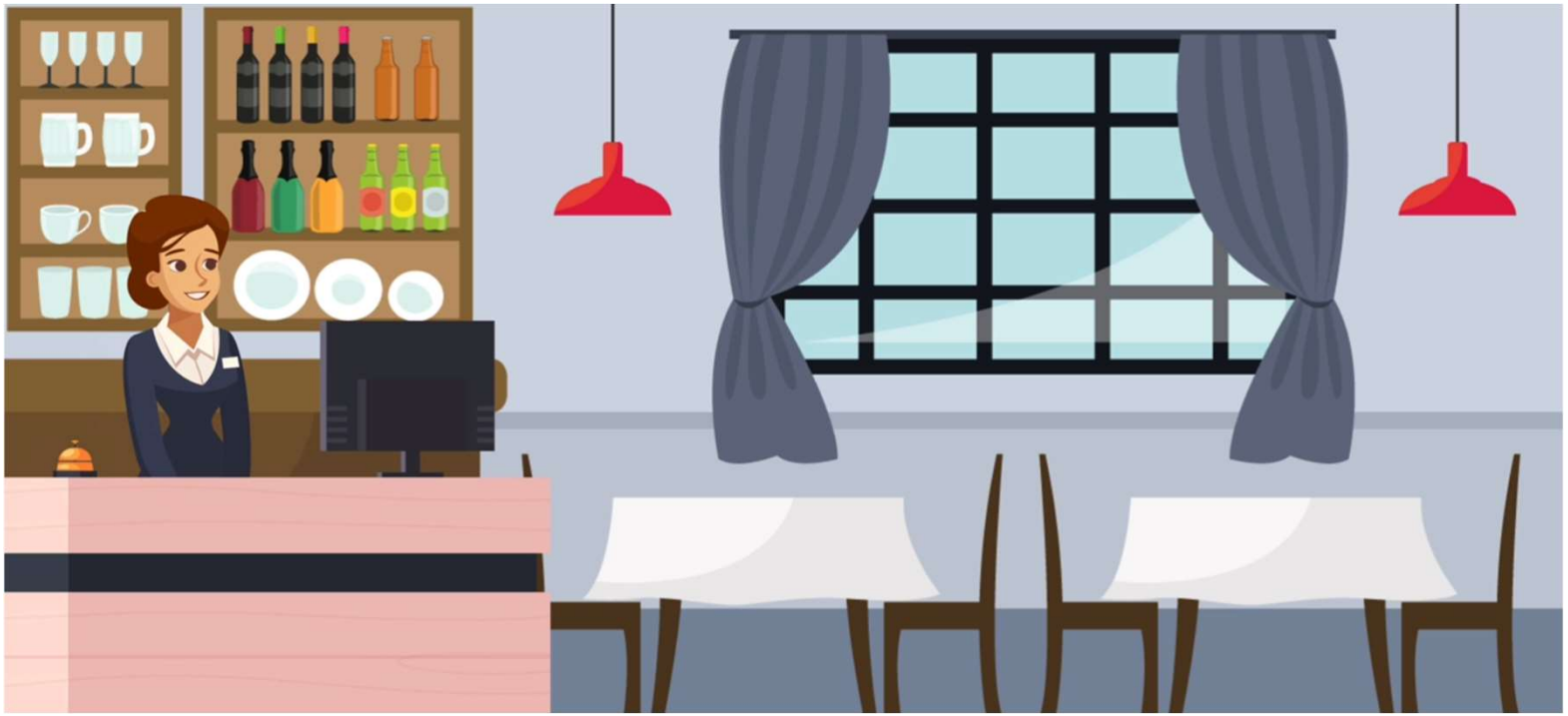


Another Advantage: Lot’s of the modules are reusable

# Object Oriented Programming

Imagine you alone have been tasked to run a restaurant

# You will have to be the receptionist

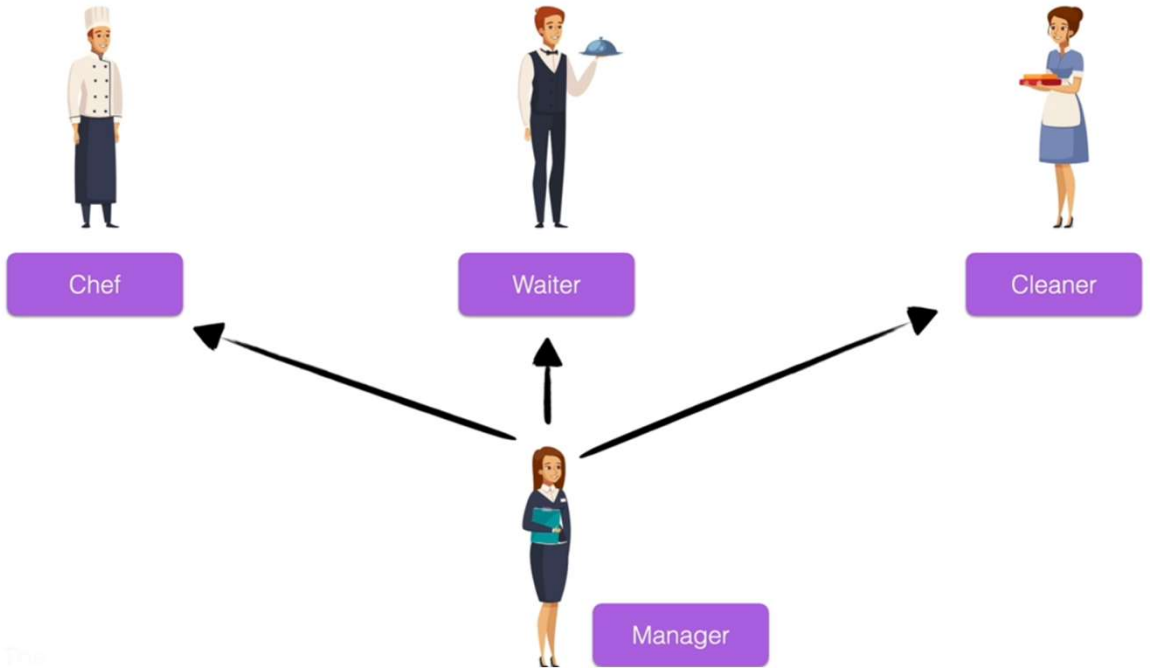


# Take Orders

# Order Needs to be cooked

# Serve the Order

# Just Guide the Hired People, They already know their work..



# Let's Model a Waiter

Waiter

**has:**

```
is_holding_plate = True
tables_responsible = [4, 5, 6]
```

**does:**

```
def take_order(table, order):
    #takes order to chef

def take_payment(amount):
    #add money to restaurant
```

# Let's Model a Waiter

**attributes:**

```
is_holding_plate = True
tables_responsible = [4, 5, 6]
```

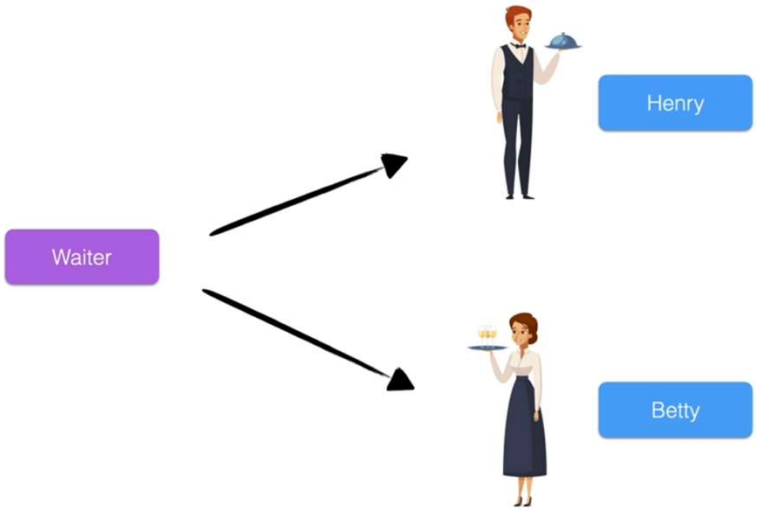
**methods:**

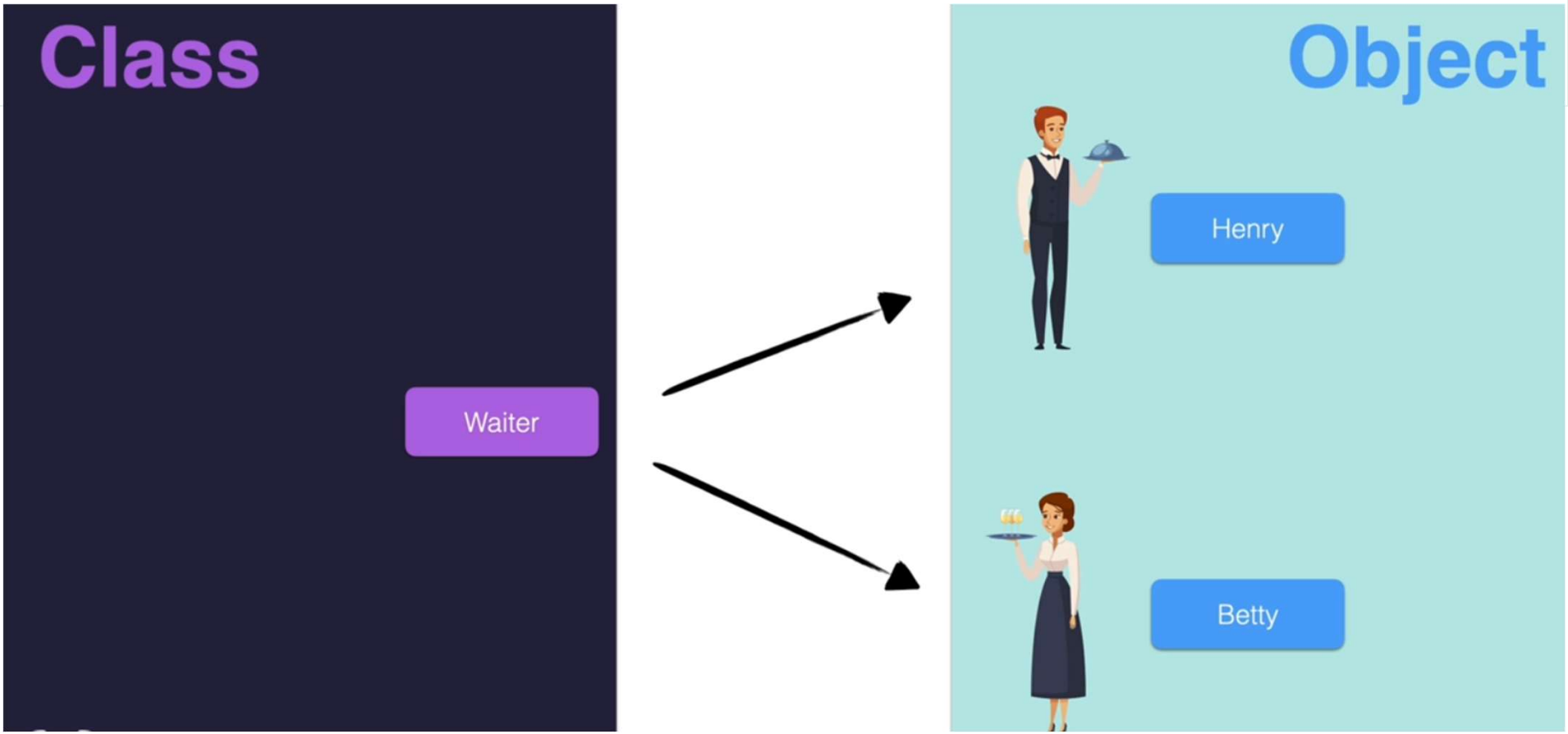
```
def take_order(table, order):
    #takes order to chef

def take_payment(amount):
    #add money to restaurant
```

After Modeling the job of waiter, we can generate as many blueprints as we want

- We call these objects




Class
Waiter
Object
Henry
Betty

# Constructing Object

# Let’s work with Turtle

- from turtle import Turtle, Screen
- obj1= turtle()
- my_screen=Screen()
- print(my_screen.canvheight)
- my_screen.exitonclick()

# Let’s work with Turtle

- obj1.shape(“turtle”)

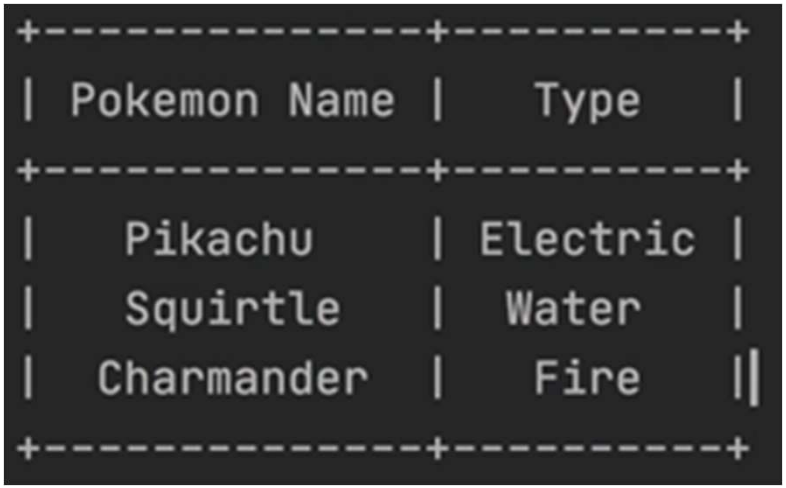
# Let’s work with Turtle

- obj1.color(“coral”)

# Let’s work with Turtle

- obj1.forward(100)

# Let’s add a package called Prettytable



# Creating table object

- from prettytable import PrettyTable
- table=PrettyTable()

# Adding a Column

- table.add_column(“Pokemon Name”,[“Pickachu”,”Squirtle”,”Charmender”])

- table.add_column(“Type”,[“Electric”,”Fire”,”Water”])

# Task

| Student Name | Class | Section | Percentage |
|---|---|---|---|
| Leanord | X | B | 91.2% |
| Penny | X | C | 63.5% |
| Howard | X | A | 90.23% |
| Bernadette | X | D | 92.7% |
| Sheldon | X | A | 98.2% |
| Raj | X | B | 88.1% |
| Amy | X | B | 95.0% |

# Creating a class

- A class is created using **class** keyword followed by a name (Identifier), finally a colon.

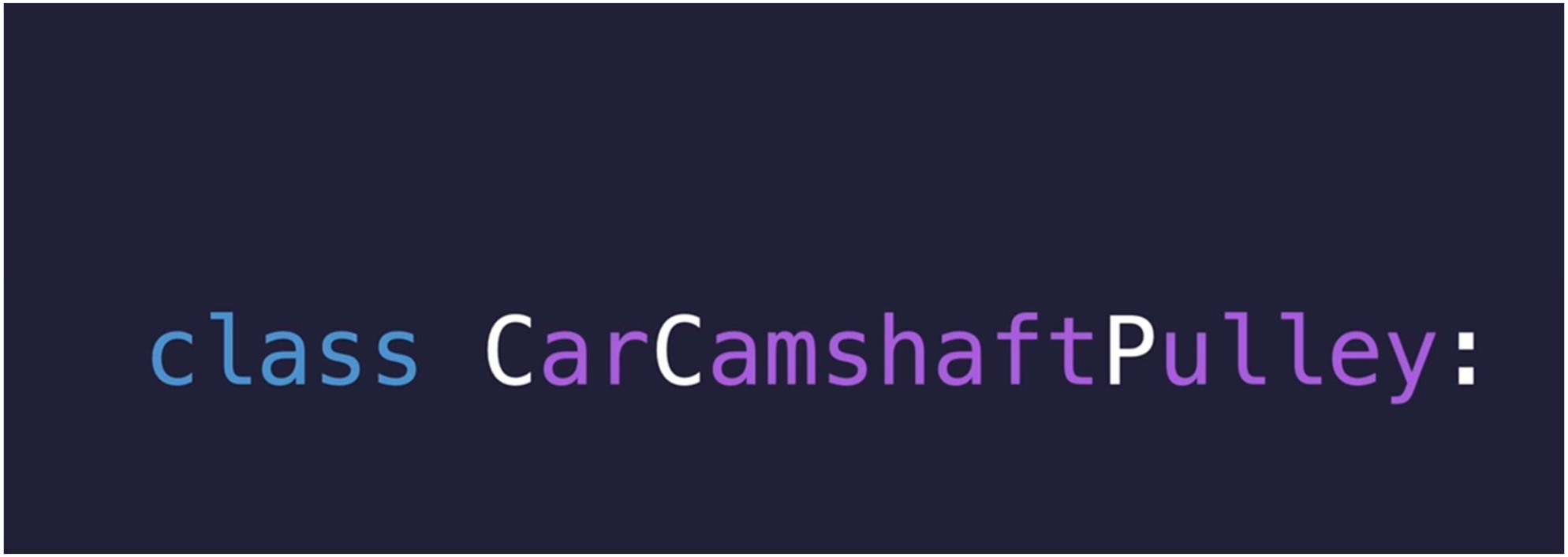
- class User:

# Usage of pass keyword

- If you define a class or a function
  - It is required that you define something in indentation

- If you want to define an empty class or function, use **pass** keyword

# Class name uses Pascal Case



# Cases

# Adding Attributes

```
class MyUser:  2 usages
    pass


user1=MyUser()
user1.name="Qadeer"
user1.id="ARI-F23-001"
print(user1.id)
```

# For multiple users we will have to write this code again and again

```
class MyUser:  2 usages
    pass

user1=MyUser()
user1.name="Qadeer"
user1.id="ARI-F23-001"
print(user1.id)

user2=MyUser()
user1.name="Fawad"
user1.id="ARI-F23-002"
print(user1.id)
```

# Constructor

Special method

Tells us what should happen when an object is created

# Constructor

```
class Car:
    def __init__(self):
    #initialise attributes
```

# Setting Attribute using Constructor

```
class Car:
    def __init__(self, seats):
        self.seats = seats
```

# Adding methods to the class

```
class User:

    def __init__(self, user_id, username):
        self.id = user_id
        self.username = username
        self.followers = 0
        self.following = 0

    def follow(self, user):
        user.followers += 1
        self.following += 1
```