SCIENTIFIC COMPUTING WITH PYTHON

Object Oriented Programming Concepts

Course Coordinator: Dr. R. Mariyal Jebasty
Assistant Professor,
Department of Physics
Wavoo Wajeeha College of Arts & Science
Kayalpatnam.

Course Instructors

1. Mrs. Pushpa, Assistant Professor in Physics, Wavoo Wajeeha Women's College of Arts & Science, Kayalpatnam.
2. Dr. S. Usharani , Assistant Professor in Physics, Wavoo Wajeeha Women's College of Arts & Science, Kayalpatnam.

Concept of OOP

CLASS

When you define a class, you define a blueprint for an object. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

OBJECT

This is the basic unit of object-oriented programming. That is both data and function that operate on data are bundled as a unit called an object.

ENCAPSULATION

Encapsulation is a process of binding data members (variables, properties) and member functions (methods) into a single unit. It is also a way of restricting access to certain properties or component. The best example for encapsulation is a class.

Encapsulation

a company
it can have several departments

- Production Department
- HR Department
- Marketing Department

all these departments make up a company

ABSTRACTION

It refers to, providing only essential information to the outside world and hiding their background details.
For example, a web server hides how it processes data it receives, the end user just hits the endpoints and gets the data back.

Abstraction

a mobile phone
you can do many things like
- make a call
- take pictures
- play games

it doesnt show you the inside process of how its doing the things
The implementation parts are hidden

INHERITANCE

The ability to create a new class from an existing class is called Inheritance. Using inheritance, we can create a Child class from a Parent class such that it inherits the properties and methods of the parent class and can have its own additional properties and methods.

Inheritance

dogs
they can have same colour
same name
same size

but they are not the same dog

POLYMORPHISM

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Polymorphism

a girl
she can be many things
mother
writer
student

a same person can have different roles

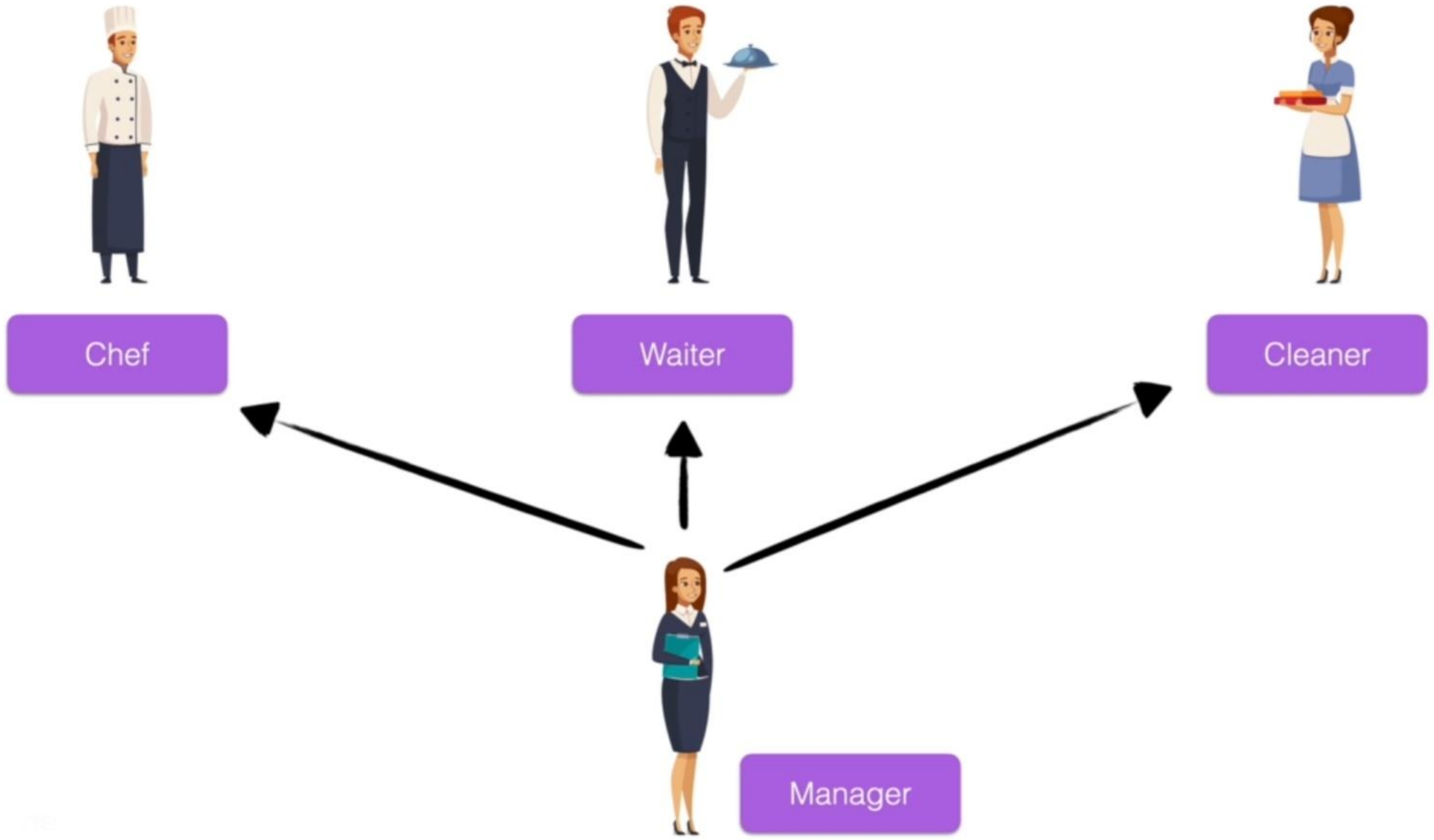
How to Use OOP

Chef

Waiter

Cleaner

Manager


Chef
Waiter
Cleaner
Manager

Waiter

has:

does:

Waiter

has:

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

does:

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
```

Waiter

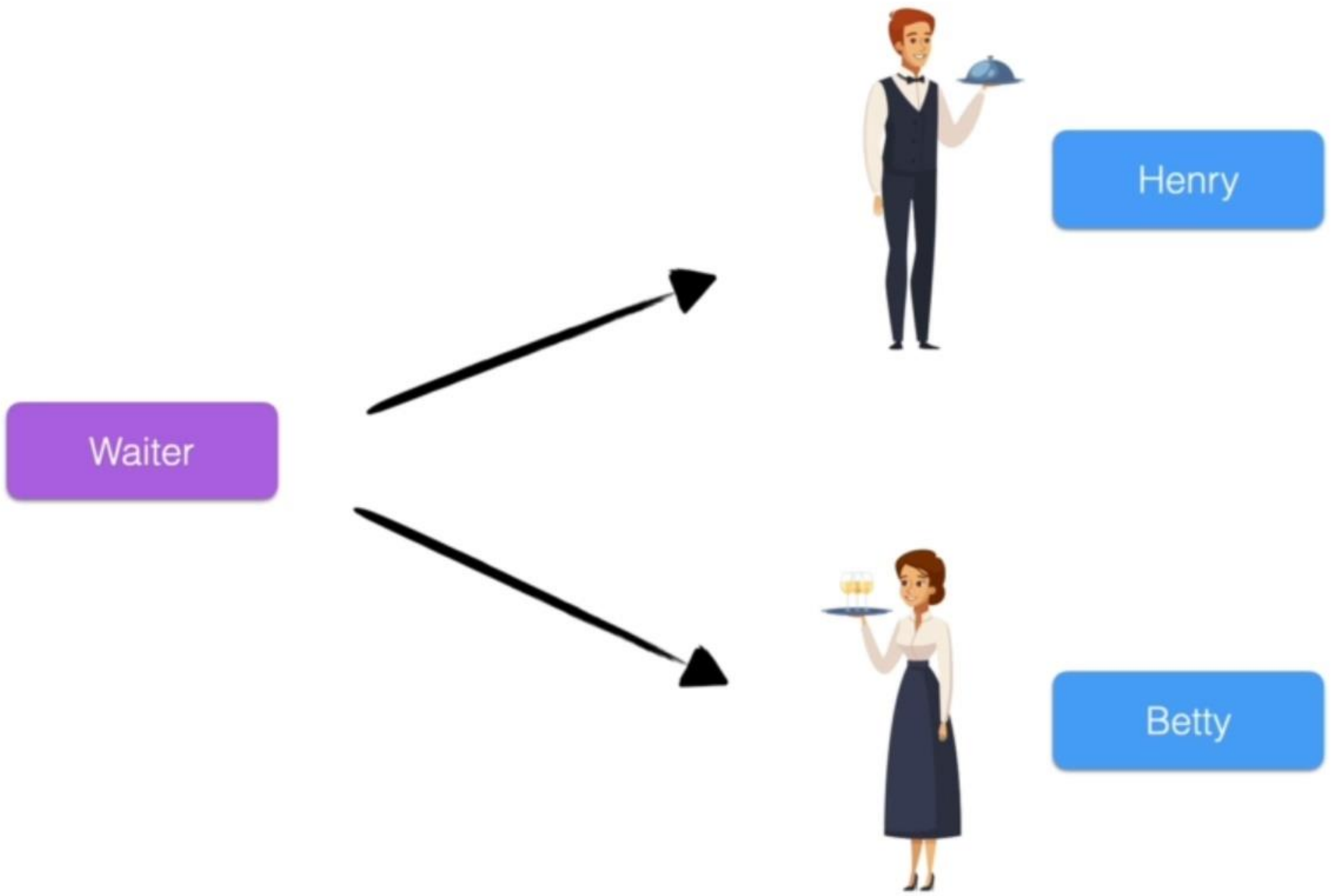
attributes:

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

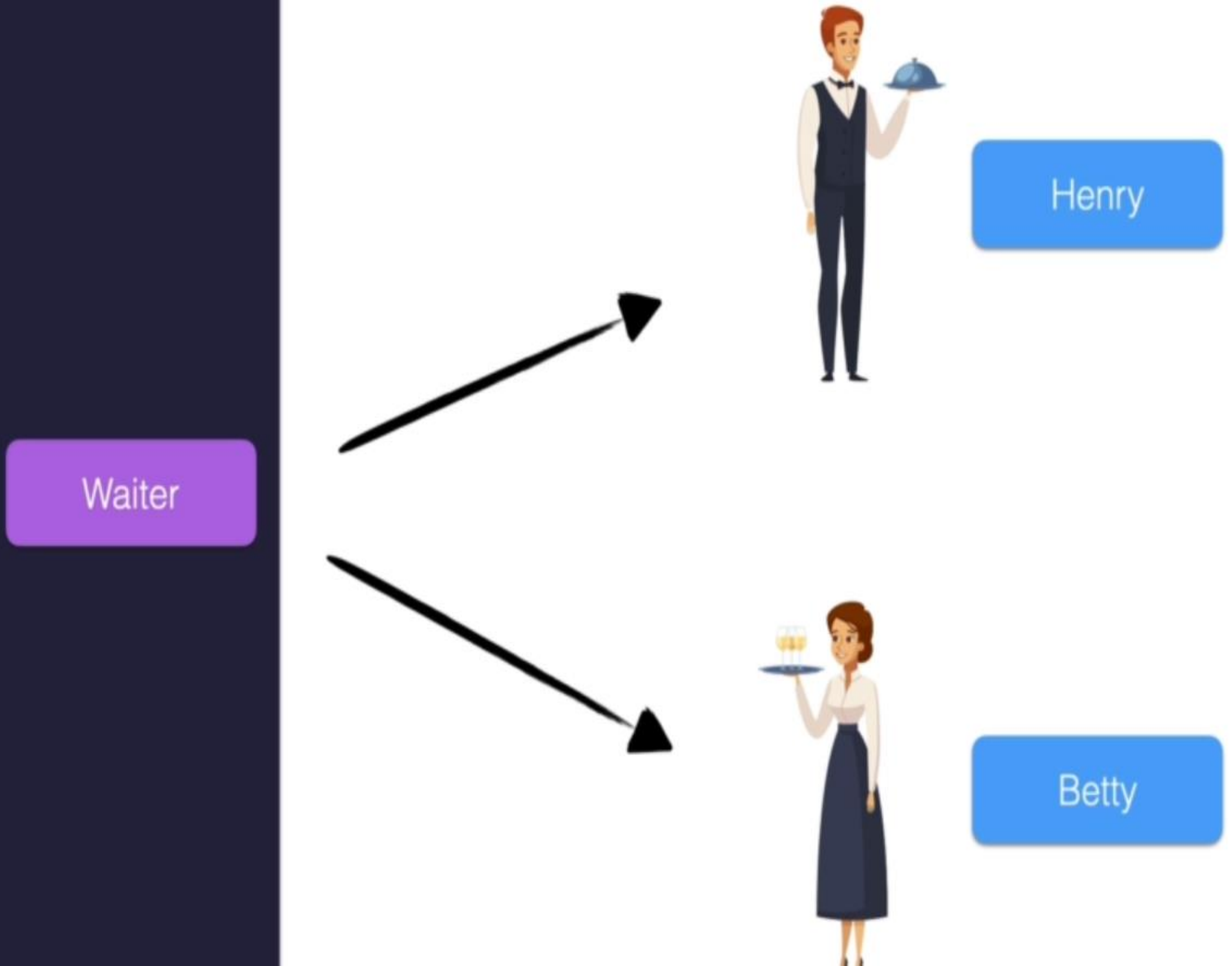
methods:

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

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

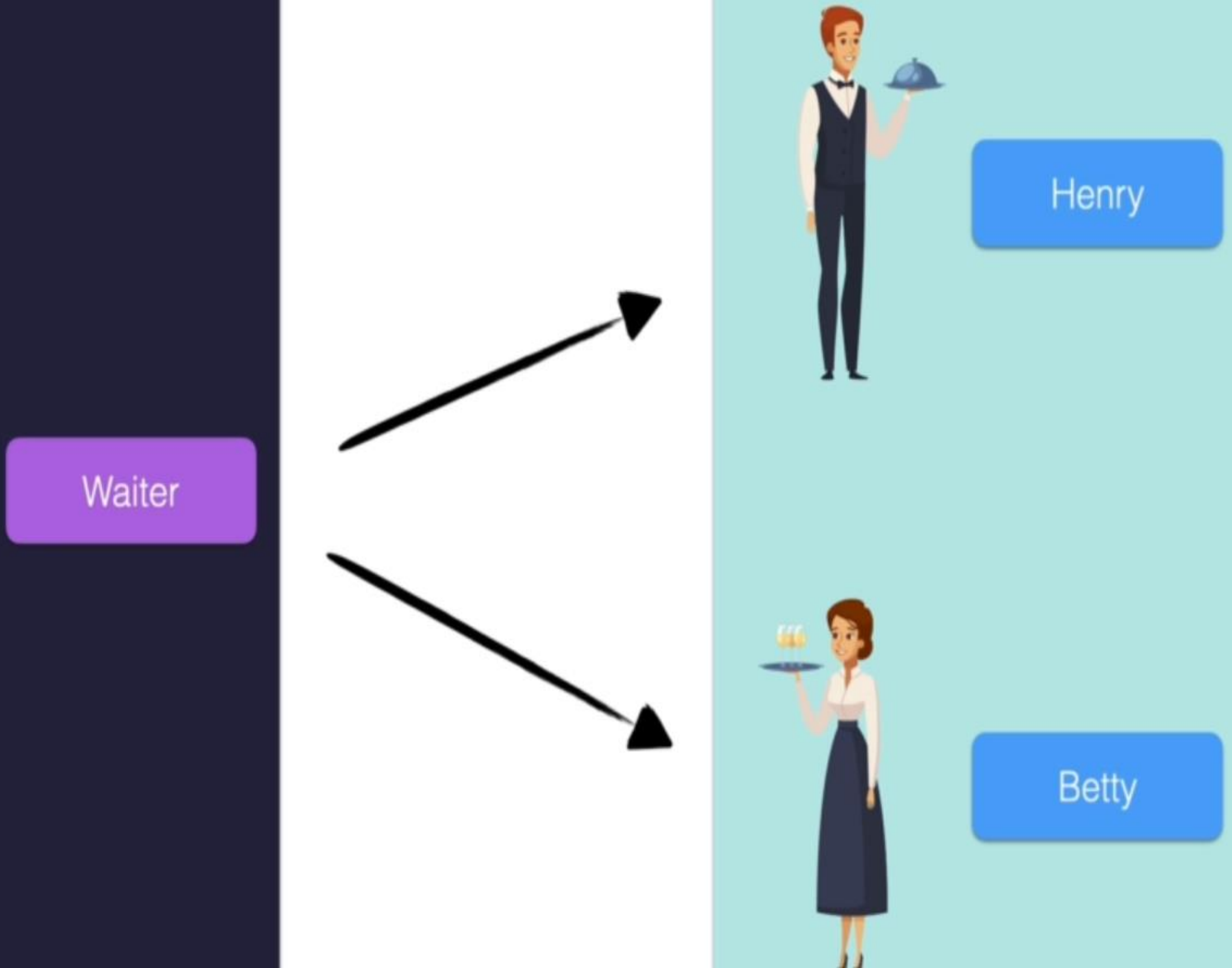

Waiter
Henry
Betty

Class



Class

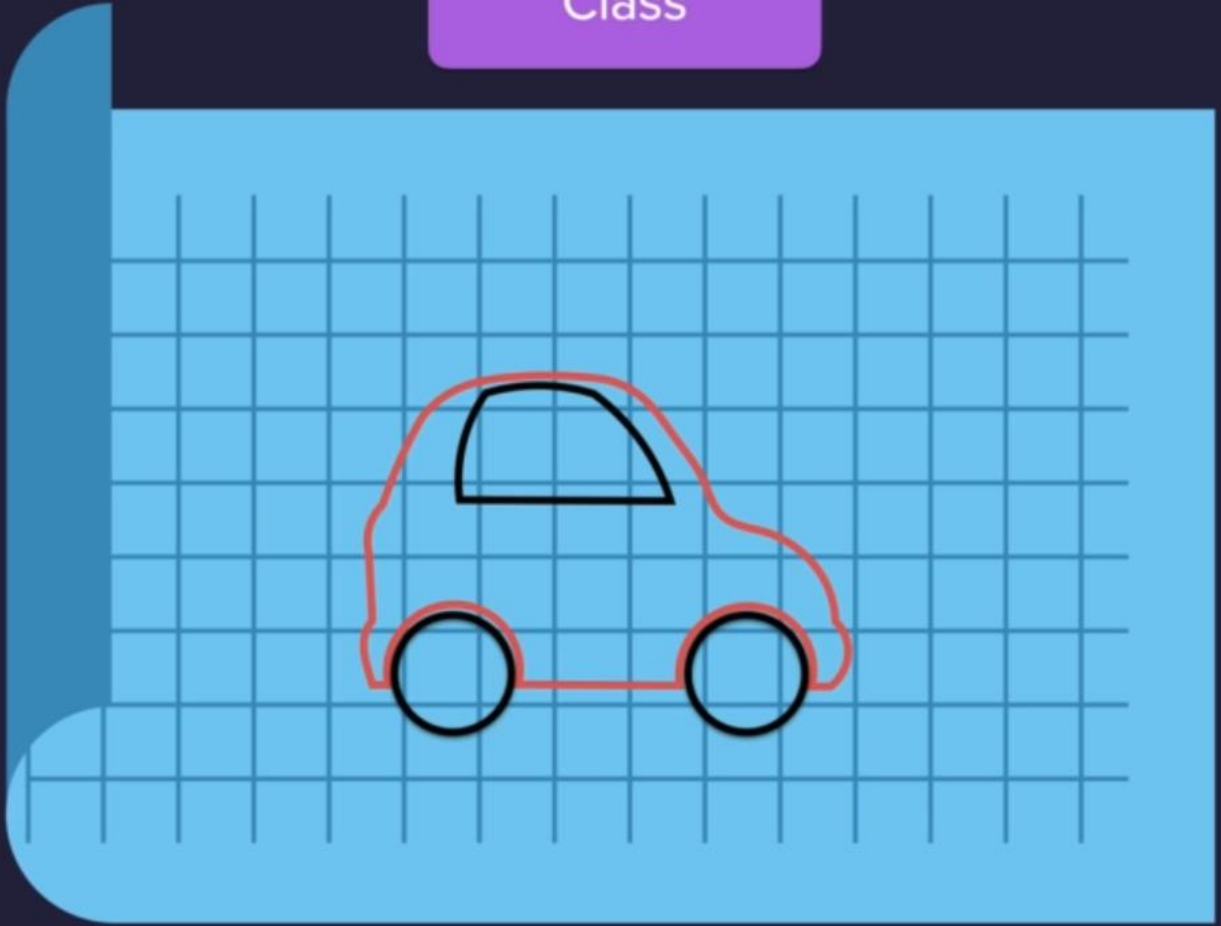
Object



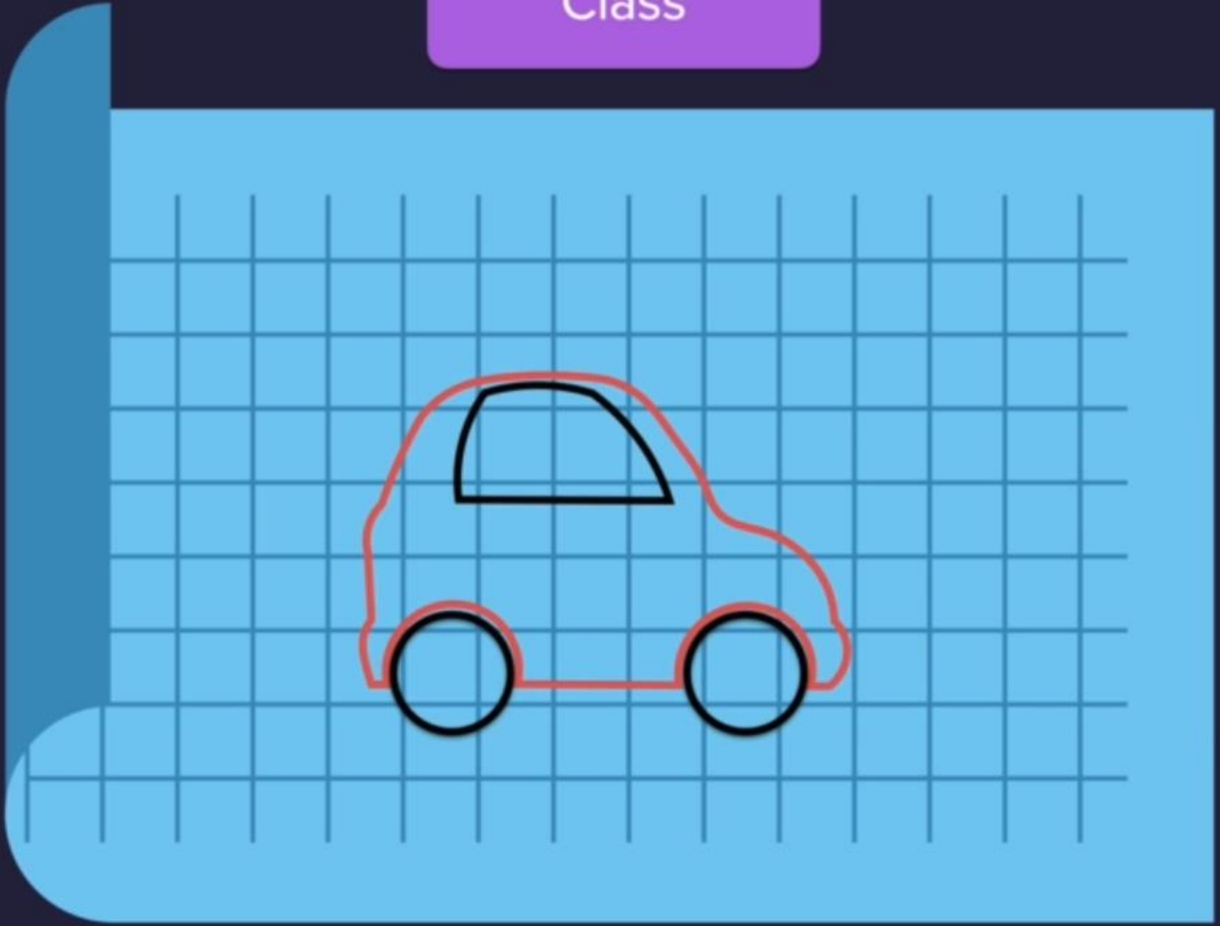
Constructing Objects

Class

Object



Class



Object

```
car = CarBlueprint()
```

Class

```
car = CarBlueprint()
```

Object

Class

```
car = CarBlueprint()
```

Car

attributes:

```
speed = 0
fuel = 32
```

```
car.speed
```

Object Attribute

```
car.speed
```

Object Methods

Car

has:

```
speed = 0
fuel = 32
```

does:

```
def move():
    speed = 60

def stop():
    speed = 0
```

Car

attributes:

```
speed = 0

fuel = 32
```

methods:

```
def move():
    speed = 60

def stop():
    speed = 0
```

```
car.stop()
```

Object Method

```
car.stop()
```

For More Details

- https://data-flair.training/blogs/python-object/
- https://data-flair.training/blogs/python-inheritance/

Object Oreiented Programming Concept

```
class Point:
    def draw(self):
        print("draw")
```

```
point = Point()
print(type(point))
print(isinstance(point, Point))
```

```
<class '__main__.Point'>
True
```

Constructor

```
# default constructor / non parameterized constructor
class Student:
    id_no = 10 # class variables
    name = 'stud1'
    def __init__(self):
        print("init method called by default")
```

```
stud = Student()
```

```
init method called by default
```

```
stud.id_no
```

```
10
```

```
Student.id_no
```

```
10
```

```
# parameterized constructor

class Person:
    def __init__(self, name, age):
        self.name = name # instance variable
        self.age = age

    def myfun(self):
        print(f"Hello my name is {self.name} and I am {self.age} years old")
```

```
p1 = Person('Jack', 20)
```

```
print(p1.name)
print(p1.age)

p1.myfun()
```

```
Jack
20
Hello my name is Jack and I am 20 years old
```

```
class MulConst:
    def __init__(self):
        print("This is first constructor")

    def __init__(self):
        print("This is second constructor")
```

```
p1 = MulConst()
```

```
This is second constructor
```

```
Student.id_no = 20
```

```
obj = Student()
```

```
init method called by default
```

```
obj.id_no
```

```
20
```

```
# magic method

class Point:

    def __init__(self, x,y):
        self.x = x
        self.y = y

    def __str__(self):
        return (f"__str__ magic function")

    def draw(self):
        print(f"Point ({self.x}, {self.y})")

point = Point(3,9)
# print(point.__str__())
print(str(point))
```

```
__str__ magic function
```

```
point.draw()
```

```
Point (3, 9)
```

```
# comparision magic methods
class Point:

    def __init__(self, x,y):
        self.x = x
        self.y = y

    def __eq__(self,other):
        return self.x == other.x and self.y == other.y

    def __lt__(self, other):
        return self.x < other.x and self.y < other.y

point = Point(3,9)
another_point = Point(13,19)

# print(point == another_point)
print(point < another_point)
```

```
True
```

```
# numeric magic method
class Point:

    def __init__(self, x,y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Point(self.x - other.x, self.y - other.y)

point = Point(3,9)
another_point = Point(13,19)

comp_add = point + another_point
print(comp_add.x, comp_add.y)

comp_sub = point - another_point
print(comp_sub.x, comp_sub.y)
```

```
16 28
-10 -10
```

```
# delete a object property

del point.y
```

```
another_point.y
```

```
    19
```

```
del another_point
```

0s completed at 5:32 PM