Classes:

This is to create our own “Datatypes”.

For instance, a string has some features, like length. A human has some features like height, weight, How annoying they are etc. These are called Attributes. They are characteristics of the object in question.

“Object” refers to one entitity belonging to the group called the “Class”.

John is an object belonging to class Human.

Duke is an object belonging to class Dog.

“Dipshit” is an object belonging to class string, and so on.

Creating classes, and maintaining attributes is a very convenient way to handle data.

```
class Human:
      def saysomething(self):
            print(“Something”)
```

the functions a human can DO are called “methods”, and they can return values, do stuff etc just like ordinary functions. The first argument of a method is always the “self”, denoting that this is a procedure any human can follow, but the person who is going to do it right now, is John.

```
X=Human()
X.saysomething()
```

That’s how a method is called. You do objectname.methodname(non self arguments)

To define the value of an attribute, we do:

```
X.name=”John”
print(X.name)
```

If you want to read the value of an attribute, there are NO brackets.

This kind of defining attributes one by one is kinda stupid, so the moment you create the object, you also assign the attributes in a compact way using something called the constructor. This is given the name initiator in python, and is denoted __init__

With double underscores on both sides for bonus fanciness.

```
class Human:
	def __init__(self,inputname,inputheight):
		self.name=inputname
		self.height=inputheight

	def saysomething(self):
		print(“Something”)
```

the self.name denotes that the object in question has acquired an attribute called name, and its value is given by the inputted value inputname. Conventionally, the input arguments, as well as the attributes, are given the same name, which kinda looks a bit weird.

now we can do:

```
X=Human(“John”,185)
```

We can still add new attributes directly to object X, by the old way, or even modify existing attributes.

Now that I’m happy with the basic things, let’s create subclasses. This is again, a tool for convenience.

Say, I now want to create an object for Spiderman, or Doctor Strange. They *do* belong to class Human, but I want to give them some special attributes, and methods. Since they do belong to class Human, they should a) have all the same attributes of a Human, and b) do anything a human can do, that is, inherit the *methods* defined for a human.

```
class Avenger(Human):
	pass
```

This presently means that Avenger, is a subclass of class Human, and everything that works for Human, works here too.

```
X=Avenger(“Bruce”,243)
```

Here, even the initiator is inherited. If we redefine it, then the new definition given inside class Avenger *overrides* the parent class’ initiator.

```
class Avenger(Human):
	def __init__(self,name,height,catchphrase):
		self.name=name
		self.height=height
		self.catchphrase= catchphrase
	def speak(self):
		print(catchphrase)
```

```
TS=Avenger(“Tony”,185,“I am Iron man.”)
TS.saysomething()
TS.speak()
```