# Ch12 - More on Inheritance + functions

- @property decorator

This can be used to convert a function look like an attribute.
@property – also called getter method.

```
class School:
    students = 2000
    newAdmission = 200
    @property ⟶ decorator
    def newSession(self):
        return self.students + self.newAdmission

new = School()
print(new.newSession) ⟶ 2200
                  └⟶ (feels like an attribute)
```

- @.setter decorator

This can be used to set some method.

```
@newSession.setter ⟶ decorator
def newSession(self, newSession):
    self.newAdmission = newsession - self.students
```

This will set the newAdmission attribute

Question → Create a class lock and create functions that are required in a lock. In the function to open a lock, make sure to validate key.
Create all required methods to work with a lock.

## Nested class

```
class House:
    class Hall:
        def TV(self):
            print("Watching TV")

new = House()
h = new.Hall()
h.TV()
```

## enumerate() function

This function adds counter to a sequence

```
for i, item in enumerate(seq):
    print(i, item)
```

## List comprehensions

```
L3 = [(i+1)*9 for i in list(range(10))]
```

This code means this:

```
L3 = []
for i in range(10):
    i += 1 (i = i+1)
    i *= 9 (i = i*9)
    L3.append(i)
```

## *args and **kwargs

These are used in functions.

→ useful in giving any number of parameters

```
def myargs(*args):          → convention
    print(args)

myargs("Python", "Java", "C")
```

To insert any other sequence syntax is like: myargs(*seq)

The * means - ALL ELEMENTS OF THE SEQUENCE

```
def cricketers(**kwargs)        → convention
    print(kwargs)

d1 = {"MS Dhoni": ["Batsman", "Wicket keeper"],
      "Yuzvendra Chahal": "Bowler",
      "Hardik Pandya": "All rounder"}

cricketers(**d1)
```

kwargs can be used in dictionaries

## Lambda functions

This is a shorter way to define functions

```
lambda parameters : operation

double = lambda a : a*2
print(double(3))
```

- getattr() function

getattr() returns the attribute of an object.

getattr(object, target attribute name)

- join() function → lets you merge string items in a list

```
a = ["Windows", "macOS", "Linux"]
b = " and ".join(a)
print(b)
```

(separator: " and ", list: a)

⇒ Windows and macOS and Linux

- ord("h")
  ↳ returns the integer representing this character.

format method

```
channel = "OceanProgrammer"
info = "Python programming with notes"
st = "This is {} who is creating {}".format(channel, info)
print(st)
```

→ This is OceanProgrammer who is creating Python programming with notes.

Here, the first {} is like {0}
the second {} is like {1}

and it continues...