Special (Magic/Dunder) Methods: this allows us to emulate some build in behavior within Python and it’s also how we implement operator overloading so what we mean of that is for example when we add two integers together and meanwhile we will add two string together then you can understand that the behavior when we add two strings together is different than when we add two integers together, so the strings were just concatenated and the integers were actually added together so depending on what object you are working with the addition actually has a different behavior .

print (1 + 2)

print (‘a’ + ‘b’)

Here we are talking about Dunder that means it is surrounded by double underscores. Basically dunder init is a special method that implicitly called when we create our employee objects here and it comes in and sets all of our attributes for us .

Apart from that there are few dunder ie \_\_repr\_\_ and \_\_str\_\_and these are implicitly called anytime we run our EPR on one of our object or str on ne of our objects

We have arithmetic dunder example:

Print(1 + 2 )

Or by using dunder we can write the same program using arithmetic dunder as

print(int.\_\_add\_\_(1,2)) lly

for string print(str.\_\_add\_\_(“a”,”b”))

Property Decorators - Getters, Setters and Deleters

Let see with example:

class Employee:

def \_\_init\_\_(self,first,last):

self.first = first

self.last = last

self.email = first + '.' + last + "@company.com"

def fullname(self):

return "{} {}".format(self.first,self.last)

emp1 = Employee("Ravi", "Reddy")

print(emp1.first)

print(emp1.email)

print(emp1.fullname())

Now the output we give as per the expected let say we are changing the first name

So emp1.firstname = ‘Sunil’ now you can find the output that the firstname has been changed to Sunil instead of Ravi but the email details are the same

Basically Property Decorator allows us to define a method but we can access it like an attribute . For example let pull the email attribute out into a method similar to full name method from above code

class Employee:

def \_\_init\_\_(self,first,last):

self.first = first

self.last = last

def email(self):

return "{} {}@company.com".format(self.first,self.last)

def fullname(self):

return "{} {}".format(self.first,self.last)

emp1 = Employee("Ravi", "Reddy")

emp1.first = "Sunil"

print(emp1.first)

print(emp1.email())

print(emp1.fullname())

But if we want to access email has an attribute rather than method we can use property method.

So we are defining our email in our class like it’s a method but we’re able to access it like an attribute and we could to this just as easily using decorator

class Employee:

def \_\_init\_\_(self,first,last):

self.first = first

self.last = last

@property

def email(self):

return "{} {}@company.com".format(self.first,self.last)

def fullname(self):

return "{} {}".format(self.first,self.last)

emp1 = Employee("Ravi", "Reddy")

emp1.first = "Sunil"

print(emp1.first)

print(emp1.email)

print(emp1.fullname())