Here is the another example for class methods to understand better . Let say for example we had someone who is using our employee class and they said hey I have these specific use cases where I’m getting employee information in a form of a string that is separated by hyphens and I’m constantly needing to parse the string before I create new employees so is there a way to just pass in a string and create an employee from that .

emp\_str\_1 = ‘John- Doe- 70000’

emp\_str\_2 = ‘Steve- Smith- 30000’

emp\_str\_3 = ‘Jane- Doe- 90000’

emp\_str\_1 = "John-Doe-70000"

emp\_str\_2 = "Steve-Smith-30000"

emp\_str\_3 = "Jane-Doe-90000"

first,last,pay = emp\_str\_1.split('-')

new\_emp1 =first,last,pay

print(new\_emp1)

Inheritance: Inheritance allow us to inherit attributes and methods from a parent class, now this is useful because we can create subclasses and get all the functionality of our parent class and then we can overwrite or add completely new functionality without affecting the parent class in any way .

Let see this by an example:

class Employee:

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

self.first = first

self.last = last

self.pay = pay

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

def fullname(self):

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

def apply\_raise(self):

self.pay = int(self.pay \* self.raise\_amt)

class Developer(Employee):

pass

dev1 = Developer("Ravi", "Reddy", 50000)

dev2 = Developer("Sunil", "Reddy",60000)

#print(Employee.fullname(dev1))

print(dev1.email)

print(dev2.email)

Here when we run the developer class it first go the class developer and in that it can find its inheriting the parent class Employee if the parent class if blank then at last it will check for builtins.object the order which it follow is called resolution order.

Suppose we want to minimize the subclass

class Employee:

raise\_amt = 1.04

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

self.first = first

self.last = last

self.pay = pay

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

def fullname(self):

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

def apply\_raise(self):

self.pay = int(self.pay \* self.raise\_amt)

class Developer(Employee):

pass

dev1 = Developer("Ravi", "Reddy", 50000)

dev2 = Developer("Sunil", "Reddy",60000)

#print(Employee.fullname(dev1))

print(dev1.email)

print(dev2.email)

print(dev1.pay)

dev1.apply\_raise()

print(dev1.pay)

Here while printing the raise amount we can find out that it is raised by 4% but suppose if we want to raise the amount by 5% we can simple do it by assigning that to a developer class ie

We can define that in Developer class

class Developer(Employee):

raise\_amt = 1.10

Note: assigning the values to one class doesn’t have the effect on the other class so it is assigned with respect to that class only.

Next let say sometimes we want to initiate our subclasses with more information than our parent class can handle .So what to I mean by that so let’s say that when we created our developers here that we wanted to also pass in their main programming language as an attribute but currently our employee class only accepts firstname, lastname, pay(self,first,last,pay)so we also wanted to pass in a programming language there then to get around this we’re going to have to give the developer class its own init method .

Note: super.init is going to pass first, last and pay to our employees init method and let that class handle those arguments .

class Employee:

raise\_amt = 1.04

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

self.first = first

self.last = last

self.pay = pay

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

def fullname(self):

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

def apply\_raise(self):

self.pay = int(self.pay \* self.raise\_amt)

class Developer(Employee):

raise\_amt = 1.10

def \_\_init\_\_(self,first,last,pay, prog\_lang):

super().\_\_init\_\_(first,last,pay)

"""Employee.\_\_init\_\_(self,first,last,pay)"""

self.prog\_lang = prog\_lang

dev1 = Developer("Ravi", "Reddy", 50000, "Pyhton")

dev2 = Developer("Sunil", "Reddy",60000, "Java")

#print(Employee.fullname(dev1))

print(dev1.email)

print(dev2.prog\_lang)

To get the more brief about it we can take another example of adding a new subclass called manager.

Basically we are checking the employees whom he is supervising.

class Manager(Employee):

def \_\_init\_\_(self,first,last,pay, employees=None):

super().\_\_init\_\_(first,last,pay)

"""Employee.\_\_init\_\_(self,first,last,pay)"""

if employees is None:

self.employees = []

else:

self.employees = employees

def add\_emp(self,emp):

if emp not in self.employees:

self.employees.append(emp)

def remove\_emp(self,emp):

if emp in self.employees:

self.employees.remove(emp)

def print\_emps(self):

for emp in self.employees:

print('-->',emp.fullname())

dev1 = Developer("Ravi", "Reddy", 50000, "Pyhton")

dev2 = Developer("Sunil", "Reddy",60000, "Java")

mgr = Manager("Kalpana", "Reddy", 40000,[dev1])

print(mgr.email)

mgr.add\_emp(dev2)

mgr.remove\_emp(dev1)

mgr.print\_emps()