```
# ------------------------------------------------
# class and static methods
# ------------------------------------------------

# class methods are mainly used as alternate constructors of the class
# first argument of the class method is class reference
# we use @classmethod built-in decorator to make a class method
# static method is just a function dropped inside a class
# for static methods, neither self is passed not cls is passed
class Employee:
    company = "Hewlett-Packard"
    def __init__(self, fname, lname, age):
        self.fname = fname
        self.lname = lname
        self.age = age

    # instance method ('self' is passed as a first argument)
    def email(self):
        print("calling email!!!!!")
        return f"{self.fname}.{self.lname}@company.com"

    # returning an instance of Employee class
    # alternate constructor
    @classmethod
    def from_string(cls, some_string):  # cls == Employee
        print(cls.company)      # Employee.company
        words = some_string.split(",")
        return cls(words[0], words[1], int(words[2]))

    # alternate constructor
    @classmethod
    def from_json(cls, json_string):
        from json import loads
        # de-serlization
        j = loads(json_string)  # converting JSON string into python data structre
        return cls(j['fname'], j['lname'], j['age'])

    @staticmethod
    def greet(name):
        return f"hello {name}"


info1 = "steve,jobs,26"
info2 = "bill,gates,28"

info3 = """{"fname": "steve",
"lname": "jobs",
"age": 28,
"phone": 2834972389,
"address": "294, bangalore",
"city": "Bangalore",
"state": "karnataka"
}
"""


e1 = Employee.from_string(info1)   # Employee.from_string(Employee, info1)
e2 = Employee.from_string(info2)    # Employee.from_string(Employee, info2)
e3 = Employee.from_json(info3)      # 3rd way of creating Employee object
e4 = Employee("john", "doe", 27)
```