

```

#-----student.py
name = input("Name: ")
house = input("House: ")
print(f"{name} from {house}")
#-----
def main():
    name = get_name()
    house = get_house()
    print(f"{name} from {house}")

def get_name():
    return input("Name: ")

def get_house():
    return input("House: ")

if __name__ == "__main__":
    main()
#=====
def main():
    name, house = get_student()
    print(f"{name} from {house}")

def get_student():
    name = input("Name: ")
    house = input("House: ")
    return name, house

if __name__ == "__main__":
    main()
#-----
class Student:
    def __init__(self, name, house):
        self.name = name
        self.house = house

def main():
    student = get_student()
    print(f"{student.name} from {student.house}")

def get_student():
    name = input("Name: ")
    house = input("House: ")
    return Student(name, house)

if __name__ == "__main__":
    main()
#-----Class
class Student: # ~~~~~~ ~~~~~~
    ...

def main():
    student = get_student()
    print(f"{student.name} from {student.house}")

def get_student():
    student = Student() # ~~~~~~ ~~~~~~
    student.name = input("Name: ") # ~~~~~~ ~~~~~~
    student.house = input("House: ")
    return student

if __name__ == "__main__":
    main()
#-----Constructor
class Student:
    def __init__(self, name, house):
        self.name = name
        self.house = house

def main():
    student = get_student()
    print(f"{student.name} from {student.house}")

```

```

def get_student():
    name = input("Name: ")
    house = input("House: ")
    return Student(name, house)

if __name__ == "__main__":
    main()
#----- raize

class Student:
    def __init__(self, name, house):
        if not name:
            raise ValueError("Missing name")
        if house not in ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]:
            raise ValueError("Invalid house")
        self.name = name
        self.house = house

def main():
    student = get_student()
    print(f"{student.name} from {student.house}")

def get_student():
    name = input("Name: ")
    house = input("House: ")
    return Student(name, house)

if __name__ == "__main__":
    main()
#----- __str__

class Student:
    def __init__(self, name, house):
        if not name:
            raise ValueError("Missing name")
        if house not in ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]:
            raise ValueError("Invalid house")
        self.name = name
        self.house = house
    def __str__(self):
        return f"{self.name} from {self.house}"

def main():
    student = get_student()
    print(student)

def get_student():
    name = input("Name: ")
    house = input("House: ")
    return Student(name, house)

if __name__ == "__main__":
    main()
#-----Decorate

class Student:
    def __init__(self, name, house):
        if not name:
            raise ValueError("Missing name")
        if house not in ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]:
            raise ValueError("Invalid house")
        self._name = name
        self._house = house
    def __str__(self):
        return f"{self._name} from {self._house}"

    @property
    def house(self):
        return self._house

    @house.setter
    def house(self, house):
        if house not in ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]:
            raise ValueError("Invalid house")
        self._house = house

```

```

def main():
    student = get_student()
    student.house="Slytherin"

    print(student)

def get_student():
    name = input("Name: ")
    house = input("House: ")
    return Student(name, house)

if __name__ == "__main__":
    main()

#----- Class method hat.py
import random
class Hat:
    def __init__(self):
        self.houses = ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]

    def sort(self, name):
        print(name, "is in", random.choice(self.houses))

hat = Hat()
hat.sort("Harry")
#-----
import random
class Hat:
    houses = ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]

    @classmethod
    def sort(cls, name):
        print(name, "is in", random.choice(cls.houses))

Hat.sort("Harry")
#----- student.py with class
class Student:
    def __init__(self, name, house):
        self.name = name
        self.house = house

    def __str__(self):
        return f"{self.name} from {self.house}"

    @classmethod
    def get(cls):
        name = input("Name: ")
        house = input("House: ")
        return cls(name, house)

def main():
    student = Student.get()
    print(student)

if __name__ == "__main__":
    main()

#----- Inheretance wizard.py
class Wizard:
    def __init__(self, name):
        if not name:
            raise ValueError("Missing name")
        self.name = name

class Student(Wizard):
    def __init__(self, name, house):
        super().__init__(name)
        self.house = house

class Professor(Wizard):
    def __init__(self, name , subject):
        super().__init__(name)

```

```
        self.subject = subject

wizard = Wizard("Albus")
student = Student("Harry", "Gryffindor")
professor = Professor("Severus", "Defense Against the Dark Arits")

print(student.name)

#-----operator overloading
class Vault:
    def __init__(self, galleons=0, sickles=0, knuts=0):
        self.galleons = galleons
        self.sickles = sickles
        self.knuts = knuts
    def __str__(self):
        return f"{self.galleons} Galleons, {self.sickles} Sickles, {self.knuts} Rnuts"

    def __add__(self, other):
        galleons = self.galleons + other.galleons
        sickles = self.sickles + other.sickles
        knuts = self.knuts + other.knuts
        return Vault(galleons, sickles, knuts)

potter = Vault(100, 50, 25)
print(potter)

weasley = Vault(25, 50, 100)
print(weasley)

total= potter + weasley
print(total)
```