student0.py

```
# Represents a student with multiple variables

name = input("Name: ")
house = input("House: ")
print(f"{name} from {house}")
```

student1.py

```python
# Modularizes getting student's name and 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()
```

student2.py

```python
# Returns student as tuple, unpacking it


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()
```

student3.py

```python
# Returns student as tuple, without unpacking it


def main():
    student = get_student()
    print(f"{student[0]} from {student[1]}")


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


if __name__ == "__main__":
    main()
```

student4.py

```
# Demonstrates immutability of tuples, removes parentheses
# https://scifi.stackexchange.com/q/105992


def main():
    student = get_student()
    if student[0] == "Padma":
        student[1] = "Ravenclaw"
    print(f"{student[0]} from {student[1]}")


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


if __name__ == "__main__":
    main()
```

student5.py

```python
# Stores student as (mutable) list


def main():
    student = get_student()
    if student[0] == "Padma":
        student[1] = "Ravenclaw"
    print(f"{student[0]} from {student[1]}")


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


if __name__ == "__main__":
    main()
```

student6.py

```python
# Stores student as dict


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


def get_student():
    student = {}
    student["name"] = input("Name: ")
    student["house"] = input("House: ")
    return student


if __name__ == "__main__":
    main()
```

student7.py

```python
# Eliminates unneeded variable


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


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


if __name__ == "__main__":
    main()
```

student8.py

```python
# Demonstrates mutability of dicts


def main():
    student = get_student()
    if student["name"] == "Padma":
        student["house"] = "Ravenclaw"
    print(f"{student['name']} from {student['house']}")


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


if __name__ == "__main__":
    main()
```

student9.py

```
# Defines class for a student


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()
```

student10.py

```python
# Adds __init__


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: ")
    student = Student(name, house)
    return student


```

student10.py

```
22  if __name__ == "__main__":
23      main()
```

student11.py

```python
# Eliminates unneeded variable


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()
```

student11.py

student12.py

```python
# Adds validation in __init__ using raise


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)
```

```
23
24
25  if __name__ == "__main__":
26      main()
```

student13.py

```
# Prints student without __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 main():
    student = get_student()
    print(student)


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

```
23
24
25  if __name__ == "__main__":
26      main()
```

student14.py

```
# Adds __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():
```

student14.py

```
23      name = input("Name: ")
24      house = input("House: ")
25      return Student(name, house)
26
27
28  if __name__ == "__main__":
29      main()
```

student15.py

```python
# Prompts for patronus too, but doesn't display yet


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

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


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


```

student15.py

```
23  def get_student():
24      name = input("Name: ")
25      house = input("House: ")
26      patronus = input("Patronus: ")
27      return Student(name, house, patronus)
28
29
30  if __name__ == "__main__":
31      main()
```

student16.py

```python
# Adds charm method to cast a charm


class Student:
    def __init__(self, name, house, patronus=None):
        if not name:
            raise ValueError("Missing name")
        if house not in ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]:
            raise ValueError("Invalid house")
        if patronus and patronus not in ["Stag", "Otter", "Jack Russell terrier"]:
            raise ValueError("Invalid patronus")
        self.name = name
        self.house = house
        self.patronus = patronus

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

    def charm(self):
        match self.patronus:
            case "Stag":
                return "🦌"
```

student16.py

```
            case "Otter":
                return "🦦"
            case "Jack Russell terrier":
                return "🐶"
            case _:
                return "🪄"


def main():
    student = get_student()
    print("Expecto Patronum!")
    print(student.charm())


def get_student():
    name = input("Name: ")
    house = input("House: ")
    patronus = input("Patronus: ") or None
    return Student(name, house, patronus)


```

student16.py

```
44  if __name__ == "__main__":
45      main()
```

student17.py

```python
# Removes patronus for simplicy, circumvents error-checking by setting attribute


class Student:
    def __init__(self, name, house):
        if not name:
            raise ValueError("Invalid 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()
    student.house = "Number Four, Privet Drive"
    print(student)


```

student17.py

```
23  def get_student():
24      name = input("Name: ")
25      house = input("House: ")
26      return Student(name, house)
27
28
29  if __name__ == "__main__":
30      main()
```

student18.py

```python
# Adds @property for house


class Student:
    def __init__(self, name, house):
        if not name:
            raise ValueError("Invalid name")
        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
```

student18.py

```
23
24
25  def main():
26      student = get_student()
27      print(student)
28
29
30  def get_student():
31      name = input("Name: ")
32      house = input("House: ")
33      return Student(name, house)
34
35
36  if __name__ == "__main__":
37      main()
```

student19.py

```python
# Adds @property for name


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

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

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, name):
        if not name:
            raise ValueError("Invalid name")
        self._name = name

    @property
```

student19.py

```
23      def house(self):
24          return self._house
25
26      @house.setter
27      def house(self, house):
28          if house not in ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]:
29              raise ValueError("Invalid house")
30          self._house = house
31
32
33  def main():
34      student = get_student()
35      print(student)
36
37
38  def get_student():
39      name = input("Name: ")
40      house = input("House: ")
41      return Student(name, house)
42
43
```

student19.py

```
44  if __name__ == "__main__":
45      main()
```

type0.py

```
1  # Prints the type of an integer
2
3  print(type(50))
```

type1.py

```
1  # Prints the type of a string
2
3  print(type("hello, world"))
```

type2.py

```
1  # Prints the type of a list
2
3  print(type([]))
```

type3.py

```
1  # Prints the type of a list
2
3  print(type(list()))
```

type4.py

```
1  # Prints the type of a dictionary
2
3  print(type({}))
```

type5.py

```
1  # Prints the type of a dictionary
2
3  print(type(dict()))
```

hat0.py

```python
# Implements sort() with an instance method

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")
```

hat1.py

```python
# Implements sort() with a class method

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")
```

student20.py

```python
# Moves get_student into Student 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)

```

```
23
24  if __name__ == "__main__":
25      main()
```

wizard.py

```
# Demonstrates inheritance [maybe don't add `if` error-checking]


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):
```

wizard.py

```
23          super().__init__(name)
24          self.subject = subject
25
26      ...
27
28
29  wizard = Wizard("Albus")
30  student = Student("Harry")
31  professor = Professor("Severus")
32  ...
```

vault0.py

```python
# Adds vaults, storing total in new vault


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} Knuts"


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

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

galleons = potter.galleons + weasley.galleons
sickles = potter.sickles + weasley.sickles
knuts = potter.knuts + weasley.knuts
```

vault0.py

```
23
24  total = Vault(galleons, sickles, knuts)
25  print(total)
```

vault1.py

```python
# Adds vaults via 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} Knuts"

    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)

```

vault1.py

```
23  weasley = Vault(25, 50, 100)
24  print(weasley)
25
26  total = potter + weasley
27  print(total)
```