(Better)

Object Oriented Programming

The Royal Edition.

Based on the OOP lecture of Harvard's CS50 course [i]

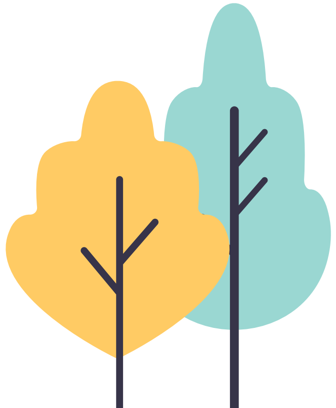
Who this?

Who this?

Het is Prins Bernhard Jr.!

- Pandjesbaas met 600 huizen in deelbezit, waarvan 100 in Amsterdam [1]
- Zit niet in de top-300 met maar 27 privé panden [2,3]
- Heeft recent 15 leuke nieuwe vertrekjes in Maastricht gekocht [4]

Bernhard heeft je hulp nodig! Hoe kan hij zijn administratie bijhouden?



Prins Bernhards wensen

Locatie
Stad van het pand

Prijs
Huurprijs van het pand

Niet te duur
Wij mogen niet te veel kosten

Mooie code!
Het liefst OOP..

Onderdelen Object Oriented Programming

01

Wat is OOP?

Alles is een object.

02

Waarom OOP?

Overzichtelijk!

03

Classes & Instances

Aanmaken van objecten.

04

Getters & Setters

Hanteren van variabelen.

05

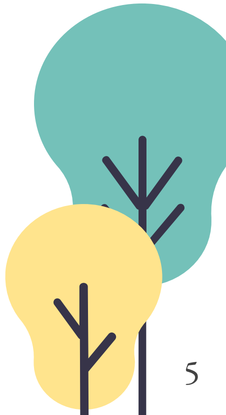
Class Methods

Functies die een instantie aanpassen.

06

Inheritance

Class 2 is een uitbreiding van Class 1.



Object Oriented Programming

Een overzichtelijke manier voor het verwerken van gelijksoortige variabelen.

Standaard:

- **Dictionaries** met vars → (locatie, prijs, oppervlak, eigenaar)
- Functies die dictionaries aanpassen

OOP:

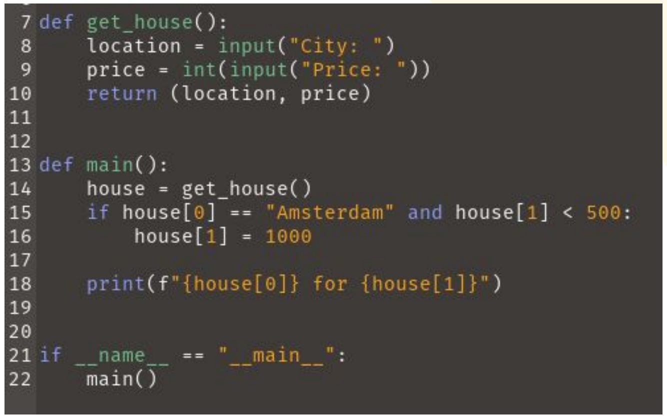
- Een algemene **Class** met de **properties**.
- **Instanties** met verschillende waarde voor properties
- Functies die op instantie niveau werken

MYHOME
REAL ESTATE

Waarom OOP?

STANDAARD

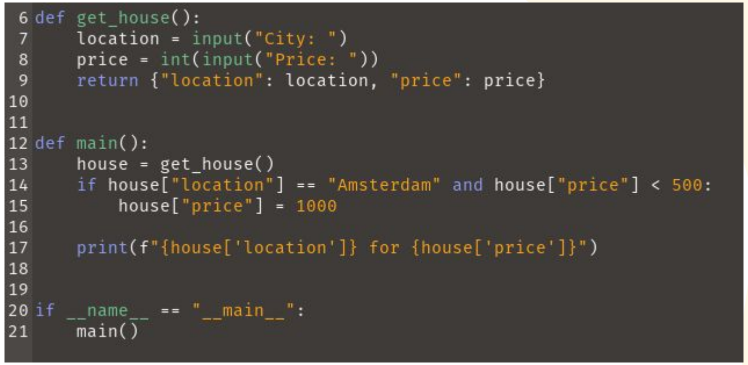
OOP


```python
def get_house():
    location = input("City: ")
    price = int(input("Price: "))
    return (location, price)


def main():
    house = get_house()
    if house[0] == "Amsterdam" and house[1] < 500:
        house[1] = 1000

    print(f"{house[0]} for {house[1]}")


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

Raises error!

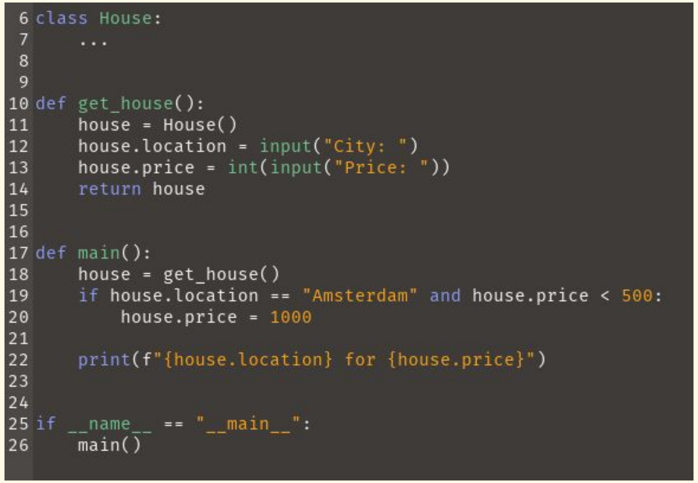
Waarom OOP?

STANDAARD



```python
def get_house():
    location = input("City: ")
    price = int(input("Price: "))
    return {"location": location, "price": price}


def main():
    house = get_house()
    if house["location"] == "Amsterdam" and house["price"] < 500:
        house["price"] = 1000

    print(f"{house['location']} for {house['price']}")


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

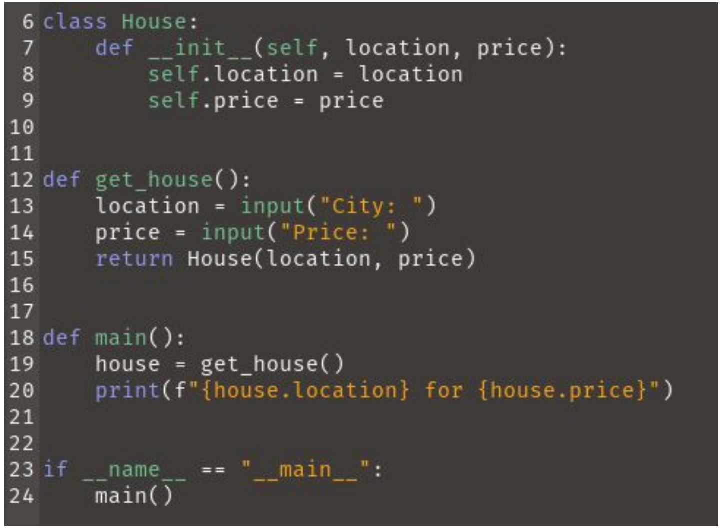
OOP

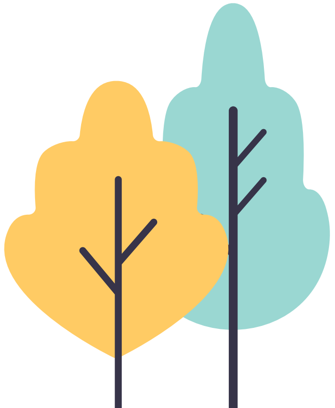


```python
class House:
    ...


def get_house():
    house = House()
    house.location = input("City: ")
    house.price = int(input("Price: "))
    return house


def main():
    house = get_house()
    if house.location == "Amsterdam" and house.price < 500:
        house.price = 1000

    print(f"{house.location} for {house.price}")


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

Classes & Instances

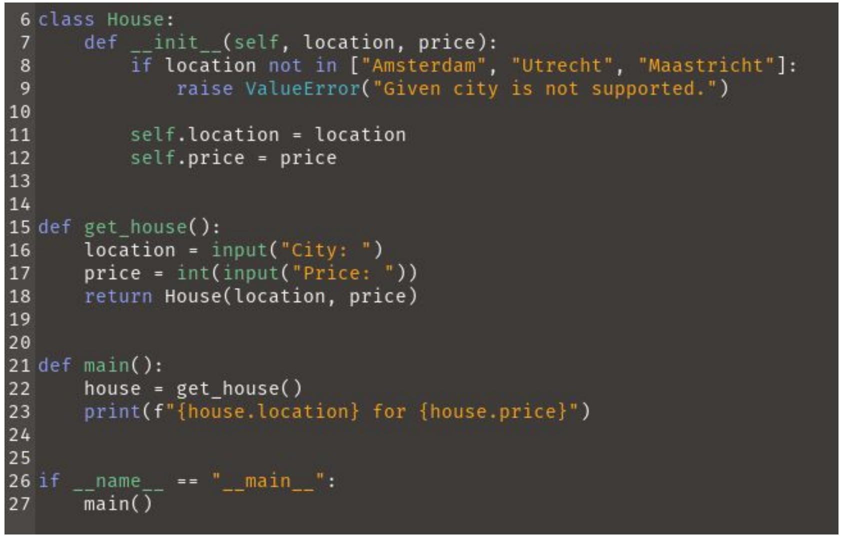
Classes zijn een definitie, een **Object** is een instantie van deze definitie.
De **__init__** functie heet ook wel de **constructor**.


```python
class House:
    def __init__(self, location, price):
        self.location = location
        self.price = price


def get_house():
    location = input("City: ")
    price = input("Price: ")
    return House(location, price)


def main():
    house = get_house()
    print(f"{house.location} for {house.price}")


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

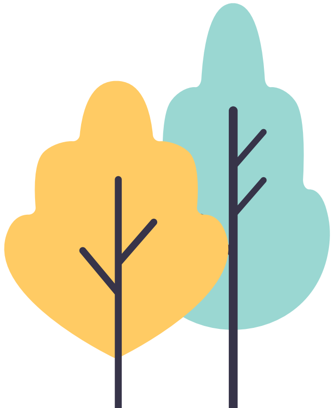


Classes & Instances

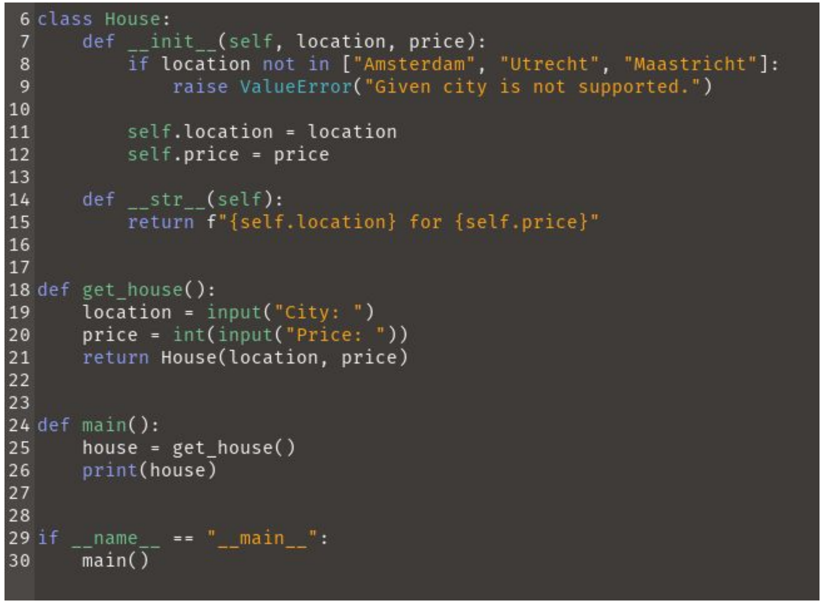
Error handling kan al tijdens het aanmaken van een object via **raise**.



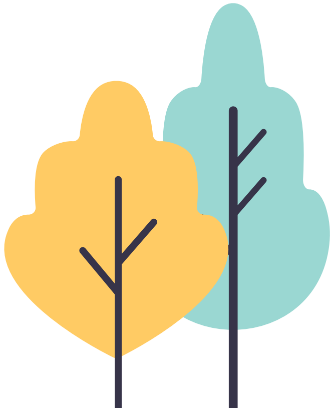
```python
class House:
    def __init__(self, location, price):
        if location not in ["Amsterdam", "Utrecht", "Maastricht"]:
            raise ValueError("Given city is not supported.")

        self.location = location
        self.price = price


def get_house():
    location = input("City: ")
    price = int(input("Price: "))
    return House(location, price)


def main():
    house = get_house()
    print(f"{house.location} for {house.price}")


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



String representatie kan aangepast worden via **__str__**.



```python
class House:
    def __init__(self, location, price):
        if location not in ["Amsterdam", "Utrecht", "Maastricht"]:
            raise ValueError("Given city is not supported.")

        self.location = location
        self.price = price

    def __str__(self):
        return f"{self.location} for {self.price}"


def get_house():
    location = input("City: ")
    price = int(input("Price: "))
    return House(location, price)


def main():
    house = get_house()
    print(house)


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



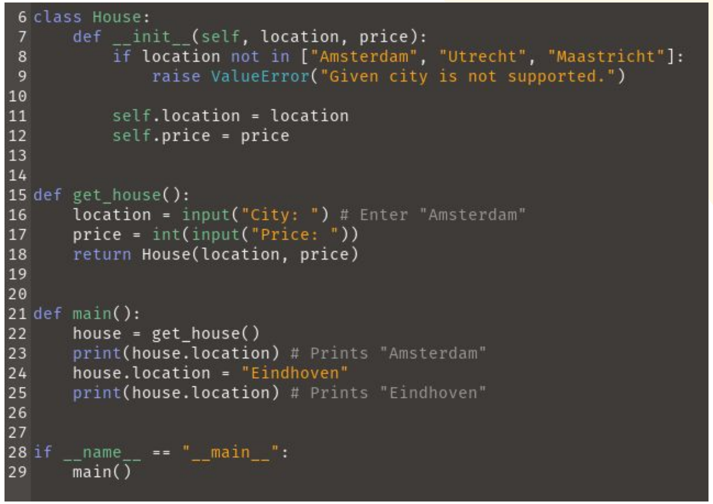
Getters & Setters

Functies binnen een Class die de properties verwerken.
Handig voor extra functionaliteit en checks!

MYHOME
REAL ESTATE

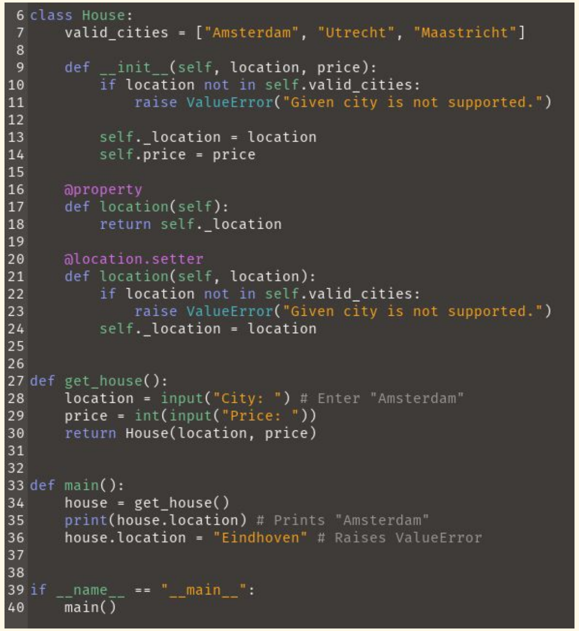
Getters & Setters

STANDAARD



```python
class House:
    def __init__(self, location, price):
        if location not in ["Amsterdam", "Utrecht", "Maastricht"]:
            raise ValueError("Given city is not supported.")

        self.location = location
        self.price = price


def get_house():
    location = input("City: ") # Enter "Amsterdam"
    price = int(input("Price: "))
    return House(location, price)


def main():
    house = get_house()
    print(house.location) # Prints "Amsterdam"
    house.location = "Eindhoven"
    print(house.location) # Prints "Eindhoven"


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

GETTERS & SETTERS



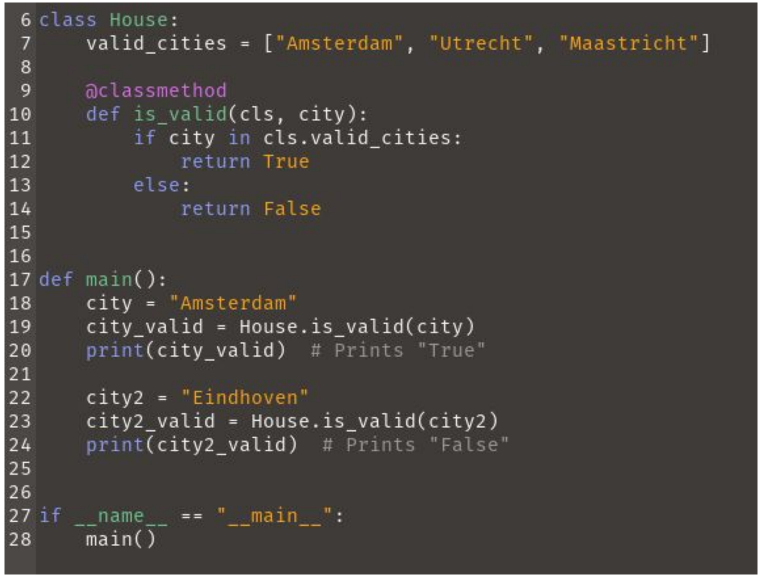
```python
class House:
    valid_cities = ["Amsterdam", "Utrecht", "Maastricht"]

    def __init__(self, location, price):
        if location not in self.valid_cities:
            raise ValueError("Given city is not supported.")

        self._location = location
        self.price = price

    @property
    def location(self):
        return self._location

    @location.setter
    def location(self, location):
        if location not in self.valid_cities:
            raise ValueError("Given city is not supported.")
        self._location = location


def get_house():
    location = input("City: ") # Enter "Amsterdam"
    price = int(input("Price: "))
    return House(location, price)


def main():
    house = get_house()
    print(house.location) # Prints "Amsterdam"
    house.location = "Eindhoven" # Raises ValueError


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

Class Methods

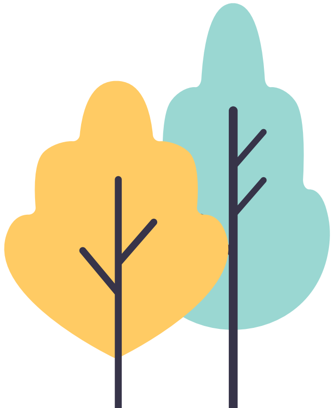
Functies binnen een Class die aangeroepen kunnen worden door een **Instantie**.

Class Methods

Class methods zijn functies die uitgevoerd kunnen worden zonder een instantie te maken.

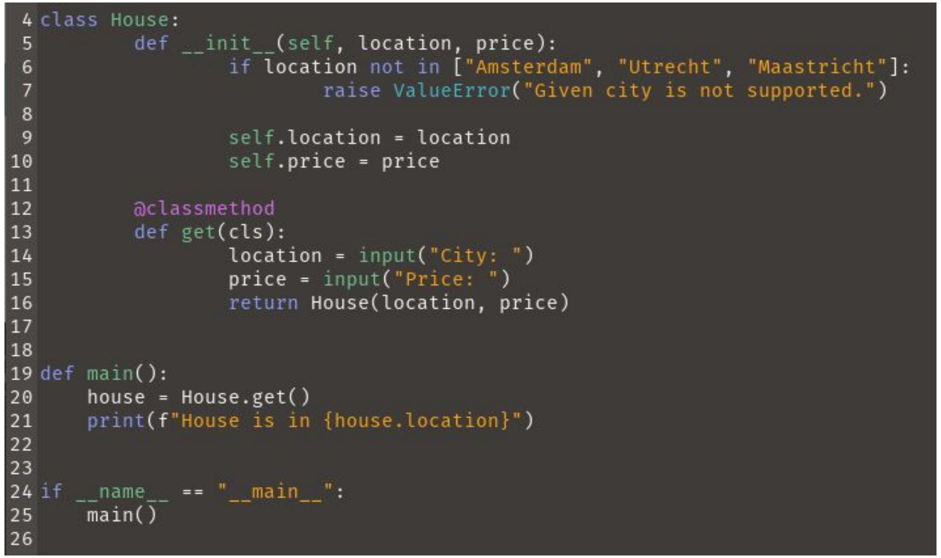


```python
class House:
    valid_cities = ["Amsterdam", "Utrecht", "Maastricht"]

    @classmethod
    def is_valid(cls, city):
        if city in cls.valid_cities:
            return True
        else:
            return False


def main():
    city = "Amsterdam"
    city_valid = House.is_valid(city)
    print(city_valid)  # Prints "True"

    city2 = "Eindhoven"
    city2_valid = House.is_valid(city2)
    print(city2_valid)  # Prints "False"


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

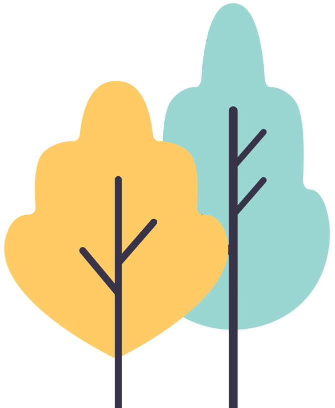


Class Methods

Class methods kunnen wel de **constructor** van de Class aanroepen.



```
class House:
        def __init__(self, location, price):
                if location not in ["Amsterdam", "Utrecht", "Maastricht"]:
                        raise ValueError("Given city is not supported.")

                self.location = location
                self.price = price

        @classmethod
        def get(cls):
                location = input("City: ")
                price = input("Price: ")
                return House(location, price)


def main():
    house = House.get()
    print(f"House is in {house.location}")


if __name__ == "__main__":
    main()

```



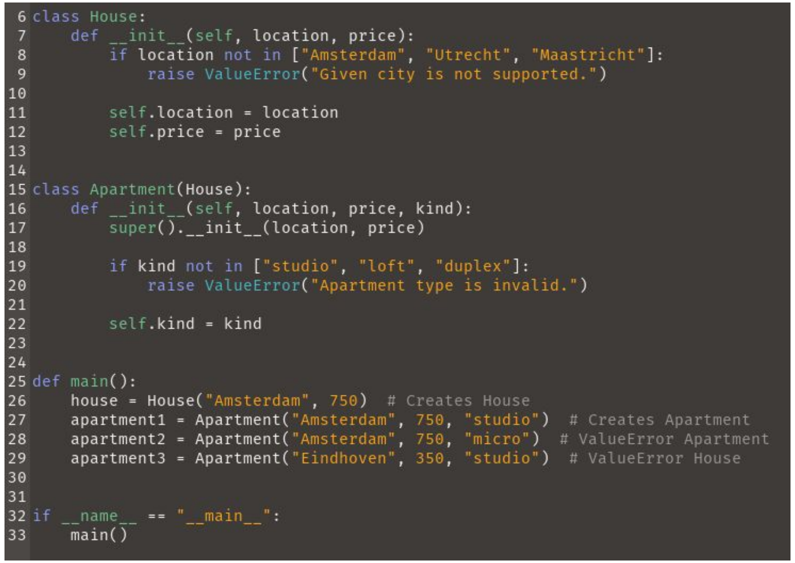
Inheritance

Een Class kan op een andere Class voortbouwen door de **properties** en **methods** te "**erven**".

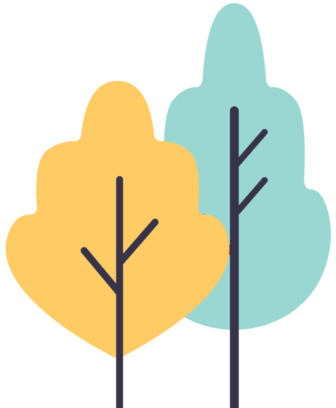
MYHOME
REAL ESTATE

Classes & Instances

Een Class (**Apartment**) kan de **properties** en **functies** van de Class **House** overnemen, en daarnaast zijn eigen definiëren.



```python
class House:
    def __init__(self, location, price):
        if location not in ["Amsterdam", "Utrecht", "Maastricht"]:
            raise ValueError("Given city is not supported.")

        self.location = location
        self.price = price


class Apartment(House):
    def __init__(self, location, price, kind):
        super().__init__(location, price)

        if kind not in ["studio", "loft", "duplex"]:
            raise ValueError("Apartment type is invalid.")

        self.kind = kind


def main():
    house = House("Amsterdam", 750)  # Creates House
    apartment1 = Apartment("Amsterdam", 750, "studio")  # Creates Apartment
    apartment2 = Apartment("Amsterdam", 750, "micro")  # ValueError Apartment
    apartment3 = Apartment("Eindhoven", 350, "studio")  # ValueError House


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



RESOURCES

[1]https://hollywoodhuizen.nl/pandjeskoning-bernhard/

[2]https://www.volkskrant.nl/mensen/waarom-prins-bernhard-niet-tot-de-grootste-verhuurders-van-nederland-behoort~b384729d/

[3]https://www.ad.nl/wonen/prins-bernhard-hoort-niet-bij-grootste-huizenbezitters~aed4a62d/

[4] https://nieuwspaal.nl/prins-bernhard-koopt-15-pandjes-tijdens-bezoek-aan-maastricht/

[i] https://cs50.harvard.edu/python/2022/notes/8/

Dat was em!

Prins Bernhard Jr. is blij!
Nog vragen?