

(Better)

# Object Oriented Programming

---

*The Royal Edition.*



Who this?



## Who this?

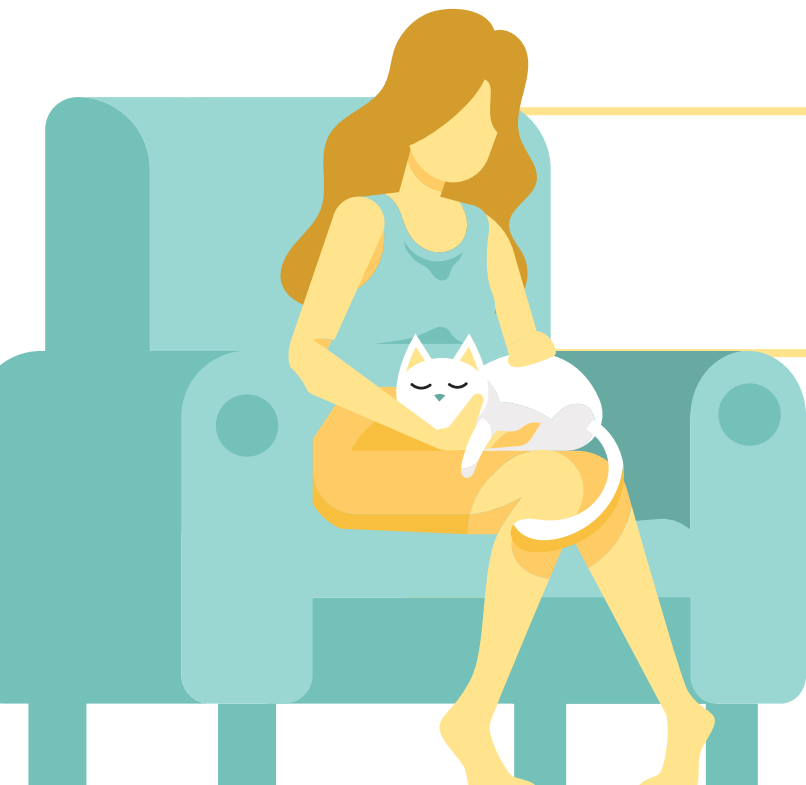


**Het is Prins Bernhard Jr.!**

- 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**

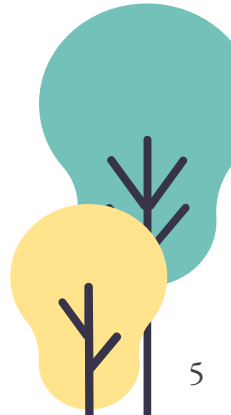
### **Class Methods**

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
- Funties die op instantie niveau werken



MYHOME  
REAL ESTATE

## Waarom OOP?

STANDAARD

OOP

```
7 def get_house():
8     location = input("City: ")
9     price = int(input("Price: "))
10    return (location, price)
11
12
13 def main():
14     house = get_house()
15     if house[0] == "Amsterdam" and house[1] < 500:
16         house[1] = 1000
17
18     print(f"{house[0]} for {house[1]}")
19
20
21 if __name__ == "__main__":
22     main()
```

Raises error!

## Waarom OOP?

### STANDAARD

```
6 def get_house():
7     location = input("City: ")
8     price = int(input("Price: "))
9     return {"location": location, "price": price}
10
11
12 def main():
13     house = get_house()
14     if house["location"] == "Amsterdam" and house["price"] < 500:
15         house["price"] = 1000
16
17     print(f"{house['location']} for {house['price']}")
18
19
20 if __name__ == "__main__":
21     main()
```

### OOP

```
6 class House:
7     ...
8
9
10 def get_house():
11     house = House()
12     house.location = input("City: ")
13     house.price = int(input("Price: "))
14     return house
15
16
17 def main():
18     house = get_house()
19     if house.location == "Amsterdam" and house.price < 500:
20         house.price = 1000
21
22     print(f"{house.location} for {house.price}")
23
24
25 if __name__ == "__main__":
26     main()
```





## Classes & Instances

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

```
6 class House:
7     def __init__(self, location, price):
8         self.location = location
9         self.price = price
10
11
12 def get_house():
13     location = input("City: ")
14     price = input("Price: ")
15     return House(location, price)
16
17
18 def main():
19     house = get_house()
20     print(f"{house.location} for {house.price}")
21
22
23 if __name__ == "__main__":
24     main()
```



## Classes & Instances

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

```
6 class House:
7     def __init__(self, location, price):
8         if location not in ["Amsterdam", "Utrecht", "Maastricht"]:
9             raise ValueError("Given city is not supported.")
10
11         self.location = location
12         self.price = price
13
14
15 def get_house():
16     location = input("City: ")
17     price = int(input("Price: "))
18     return House(location, price)
19
20
21 def main():
22     house = get_house()
23     print(f"{house.location} for {house.price}")
24
25
26 if __name__ == "__main__":
27     main()
```



## Classes & Instances

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



```
6 class House:
7     def __init__(self, location, price):
8         if location not in ["Amsterdam", "Utrecht", "Maastricht"]:
9             raise ValueError("Given city is not supported.")
10
11         self.location = location
12         self.price = price
13
14     def __str__(self):
15         return f"{self.location} for {self.price}"
16
17
18 def get_house():
19     location = input("City: ")
20     price = int(input("Price: "))
21     return House(location, price)
22
23
24 def main():
25     house = get_house()
26     print(house)
27
28
29 if __name__ == "__main__":
30     main()
```



# Getters & Setters

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



MYHOME  
REAL ESTATE

## Getters & Setters

### STANDAARD

```
6 class House:
7     def __init__(self, location, price):
8         if location not in ["Amsterdam", "Utrecht", "Maastricht"]:
9             raise ValueError("Given city is not supported.")
10
11         self.location = location
12         self.price = price
13
14
15 def get_house():
16     location = input("City: ") # Enter "Amsterdam"
17     price = int(input("Price: "))
18     return House(location, price)
19
20
21 def main():
22     house = get_house()
23     print(house.location) # Prints "Amsterdam"
24     house.location = "Eindhoven"
25     print(house.location) # Prints "Eindhoven"
26
27
28 if __name__ == "__main__":
29     main()
```

### GETTERS & SETTERS

```
6 class House:
7     valid_cities = ["Amsterdam", "Utrecht", "Maastricht"]
8
9     def __init__(self, location, price):
10         if location not in self.valid_cities:
11             raise ValueError("Given city is not supported.")
12
13         self._location = location
14         self.price = price
15
16     @property
17     def location(self):
18         return self._location
19
20     @location.setter
21     def location(self, location):
22         if location not in self.valid_cities:
23             raise ValueError("Given city is not supported.")
24         self._location = location
25
26
27 def get_house():
28     location = input("City: ") # Enter "Amsterdam"
29     price = int(input("Price: "))
30     return House(location, price)
31
32
33 def main():
34     house = get_house()
35     print(house.location) # Prints "Amsterdam"
36     house.location = "Eindhoven" # Raises ValueError
37
38
39 if __name__ == "__main__":
40     main()
```



# Class Methods

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



MYHOME  
REAL ESTATE



## Class Methods

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



```
6 class House:
7     valid_cities = ["Amsterdam", "Utrecht", "Maastricht"]
8
9     @classmethod
10    def is_valid(cls, city):
11        if city in cls.valid_cities:
12            return True
13        else:
14            return False
15
16
17 def main():
18     city = "Amsterdam"
19     city_valid = House.is_valid(city)
20     print(city_valid) # Prints "True"
21
22     city2 = "Eindhoven"
23     city2_valid = House.is_valid(city2)
24     print(city2_valid) # Prints "False"
25
26
27 if __name__ == "__main__":
28     main()
```



## Class Methods

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

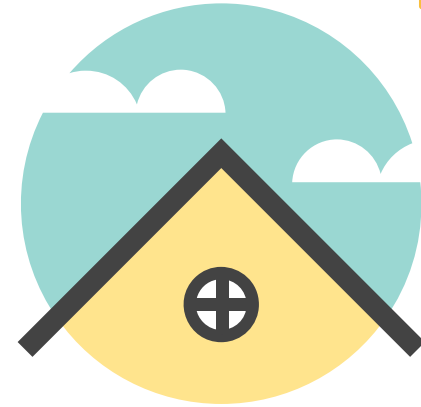


```
4 class House:
5     def __init__(self, location, price):
6         if location not in ["Amsterdam", "Utrecht", "Maastricht"]:
7             raise ValueError("Given city is not supported.")
8
9         self.location = location
10        self.price = price
11
12        @classmethod
13        def get(cls):
14            location = input("City: ")
15            price = input("Price: ")
16            return House(location, price)
17
18
19 def main():
20     house = House.get()
21     print(f"House is in {house.location}")
22
23
24 if __name__ == "__main__":
25     main()
26
```



# 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.



```
6 class House:
7     def __init__(self, location, price):
8         if location not in ["Amsterdam", "Utrecht", "Maastricht"]:
9             raise ValueError("Given city is not supported.")
10
11         self.location = location
12         self.price = price
13
14
15 class Apartment(House):
16     def __init__(self, location, price, kind):
17         super().__init__(location, price)
18
19         if kind not in ["studio", "loft", "duplex"]:
20             raise ValueError("Apartment type is invalid.")
21
22         self.kind = kind
23
24
25 def main():
26     house = House("Amsterdam", 750) # Creates House
27     apartment1 = Apartment("Amsterdam", 750, "studio") # Creates Apartment
28     apartment2 = Apartment("Amsterdam", 750, "micro") # ValueError Apartment
29     apartment3 = Apartment("Eindhoven", 350, "studio") # ValueError House
30
31
32 if __name__ == "__main__":
33     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?

CREDITS: This presentation template was created by **Slidesgo**,  
including icons by **Flaticon**, and infographics & images by **Freepik**.

**Please keep this slide for attribution.**

