

(Better)

Object Oriented Programming

The Royal Edition.



Who this?



Who this?

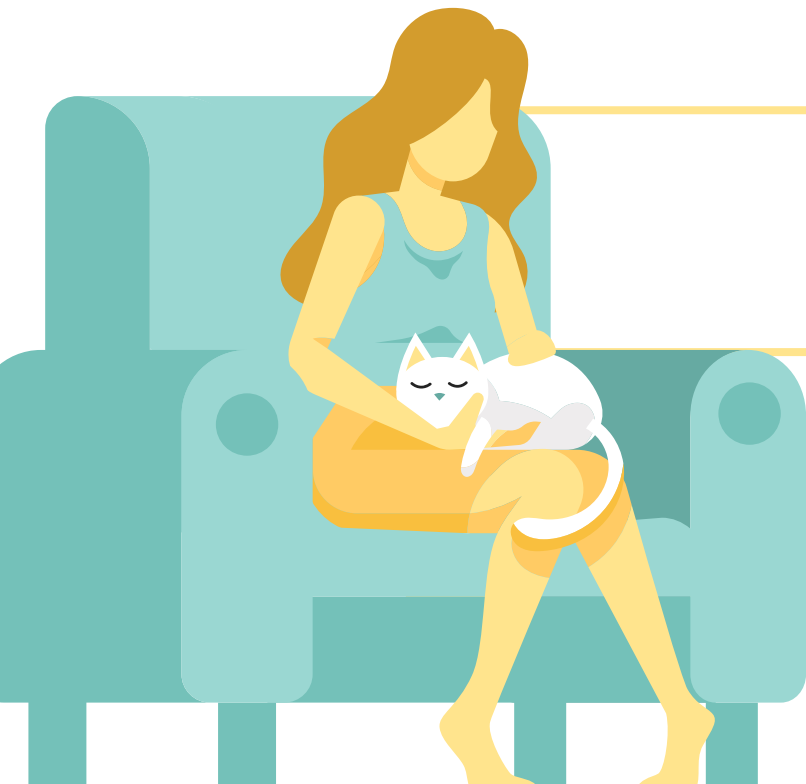


Het is Prins Bernhard!

- Huisjesmelker van het jaar prijswinnaar in 2017 [1]
- 2021: bezit 600 huizen in Nederland, waarvan 100 in Amsterdam [2]
- Heeft recent 15 leuke nieuwe vertrekjes in Maastricht gekocht [3]

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

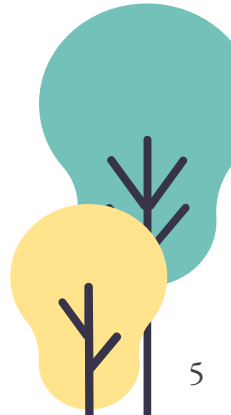
Class Methods

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



Waarom OOP?

STANDAARD

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

Raises error!

OOP

```
4 class House:
5     ...
6
7
8 def get_house():
9     house = House()
10    house.location = input("City: ")
11    house.price = input("Price: ")
12    return house
13
14
15 def main():
16    house = get_house()
17    if house.location == "Amsterdam" and house.price < 500:
18        house.price = 1000
19
20    print(f"{house.location} for {house.price}")
21
22
23 if __name__ == "__main__":
24    main()
25
```



Classes & Instances

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



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




Classes & Instances

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

```
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
13 def get_house():
14     location = input("City: ")
15     price = input("Price: ")
16     return House(location, price)
17
18
19 def main():
20     house = get_house()
21     print(f"{house.location} for {house.price}")
22
23
24 if __name__ == "__main__":
25     main()
26
```

String representatie kan aangepast worden via `__str__`.

```
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    def __str__(self):
13        return f"{self.location} for {self.price}"
14
15
16 def get_house():
17     location = input("City: ")
18     price = input("Price: ")
19     return House(location, price)
20
21
22 def main():
23     house = get_house()
24     print(house)
25
26
27 if __name__ == "__main__":
28     main()
29
```

Getters & Setters

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



Getters & Setters

STANDAARD

```
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
13 def get_house():
14     location = input("City: ") # Enter "Amsterdam"
15     price = input("Price: ")
16     return House(location, price)
17
18
19 def main():
20     house = get_house()
21     print(house.location) # Prints "Amsterdam"
22     house.location = "Eindhoven"
23     print(house.location) # Prints "Eindhoven"
24
25
26 if __name__ == "__main__":
27     main()
28
```

GETTERS & SETTERS

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

Class Methods

Funcies 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.

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



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".





Classes & Instances

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

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

RESOURCES

- [1] <https://www.parool.nl/nieuws/prins-bernhard-genomineerd-voor-huisjesmelker-van-het-jaar~b1c21280/>
- [2] <https://hollywoodhuizen.nl/pandjeskoning-bernhard/>
- [3] <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 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.

