PROGRAMMEREN 2

TOPIC 8: OBJECTGERICHT PROGRAMMEREN - UITBREIDING

Download PDF-versie

UCLL HOGESCHOOL

OVERZICHT

- Herhaling Programmeren 1
- Aandachtspunten
- Standaardwaarden instellen
- Klassen combineren

HERHALING PROGRAMMEREN 1

WAT IS OBJECTGERICHT PROGRAMMEREN?

PROGRAMMEERPARADIGMA'S

Doorheen de jaren zijn er meerdere paradigma's ontwikkeld om programma's te ontwerpen.

Tot nu hebben we vooral **imperatief programmeren** behandeld. Het houdt in dat je de computer stap voor stap bevelen geeft. Bijvoorbeeld:

- Geef deze variabele de waarde 5
- Roep deze functie 5 keer op
- Tel stap voor stap van 10 tot 1

Imperatief programmeren is een van de meest gebruikte paradigma's en werkt heel goed voor eenvoudige programma's (minder dan ~500 lijnen code).

Deze lessen focussen we op een ander paradigma: **objectgericht programmeren** of **OGP** (Engels: *Object Oriented Programming, OOP*).

OBJECTEN

OGP samengevat in één zin:

> *"Alles is een object."*

Het doel van OGP is om de werkelijkheid te modelleren als objecten die met elkaar interageren.

Voorbeelden van objecten:

- Een tafel, stoel, deur, bord, ...
- Een kat, hond, mens, plant, ...
- Een bedrijf, bankrekening, kleur, vak, datum, ...

Object is dus heel ruim te interpreteren.

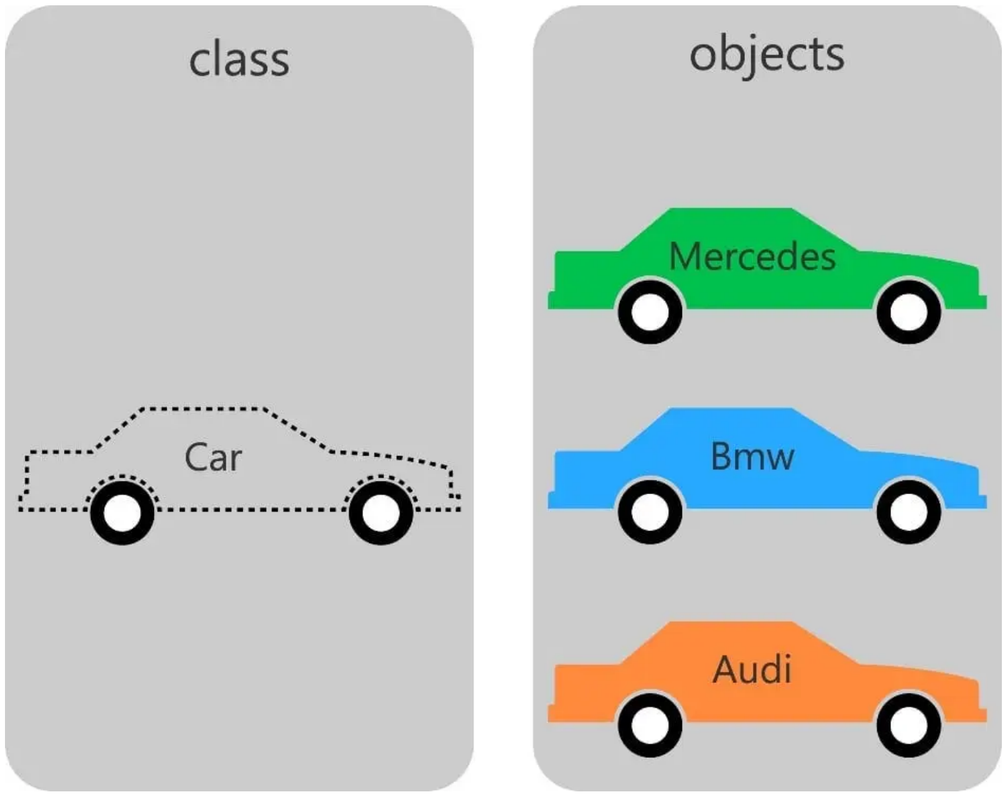
Veel objecten zullen op elkaar lijken. In OGP kan je een hele categorie van objecten één keer beschrijven, en nadien hergebruiken.

OGP: TERMINOLOGIE

Je beschrijft objecten door middel van **klassen**. Een klasse (Engels: *class*) werkt als een 'blauwafdruk' die je nadien kan gebruiken om een object te 'bouwen'.

Een object is één unieke **instantie** van een klasse. Vergelijk met een uniek gebouw dat je gecreëerd hebt op basis van de blauwafdruk. Men zegt dat men de klasse *instantieert*.

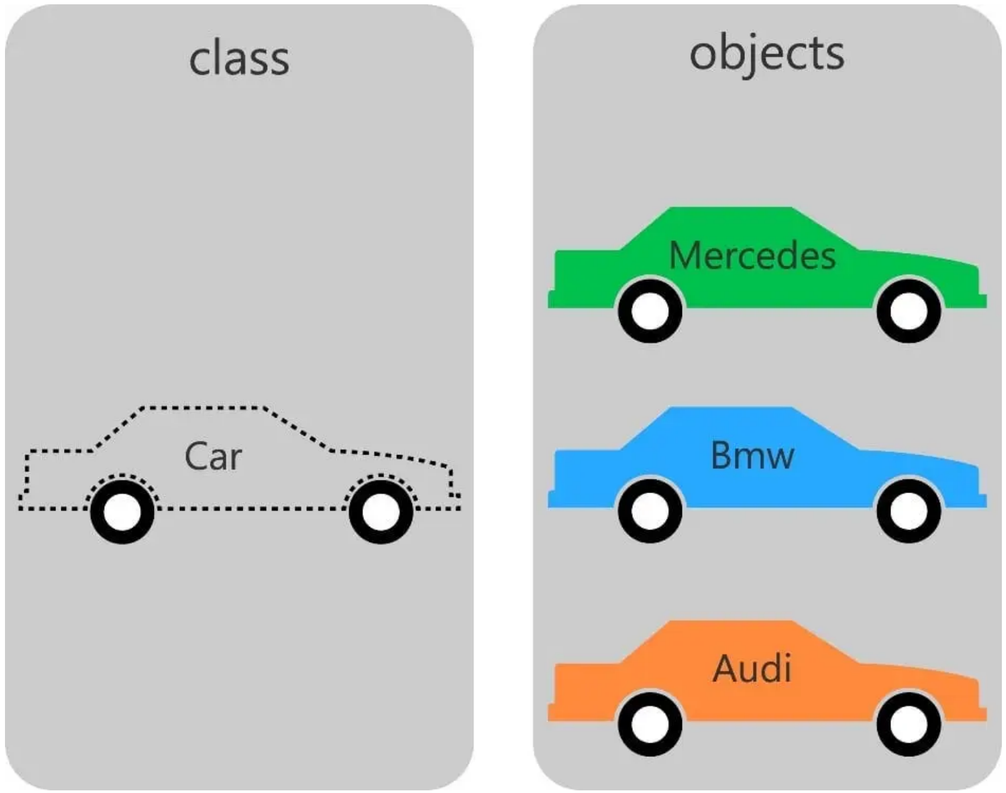
Je kan van één klasse zoveel objecten maken als nodig.



OGP: TERMINOLOGIE

De kenmerken van een klasse en haar objecten noemt men ook de **attributen** van een klasse (Engels: *attributes*). Een andere veel voorkomen benaming is **velden** (Engels: *fields*).

Welke attributen zou je toevoegen aan de klasse `Car`?



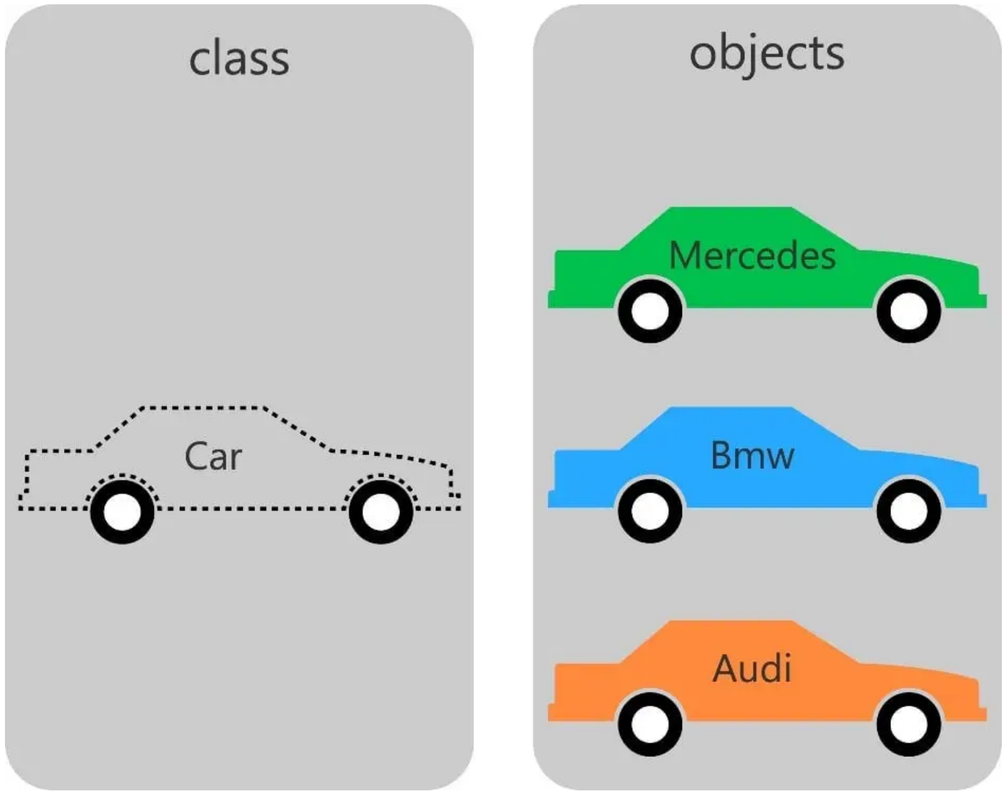
OGP: TERMINOLOGIE

De kenmerken van een klasse en haar objecten noemt men ook de **attributen** van een klasse (Engels: *attributes*). Een andere veel voorkomen benaming is **velden** (Engels: *fields*).

Welke attributen zou je toevoegen aan de klasse `Car`?

Antwoord:

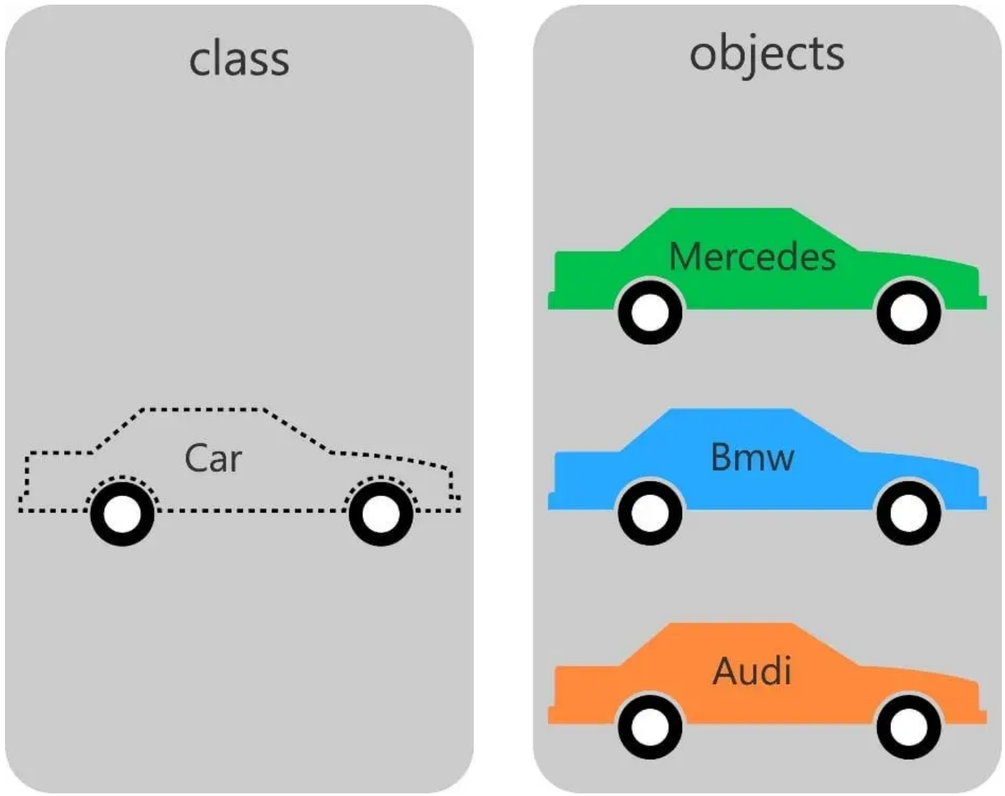
- kleur
- merk
- brandstof
- ...



OGP: TERMINOLOGIE

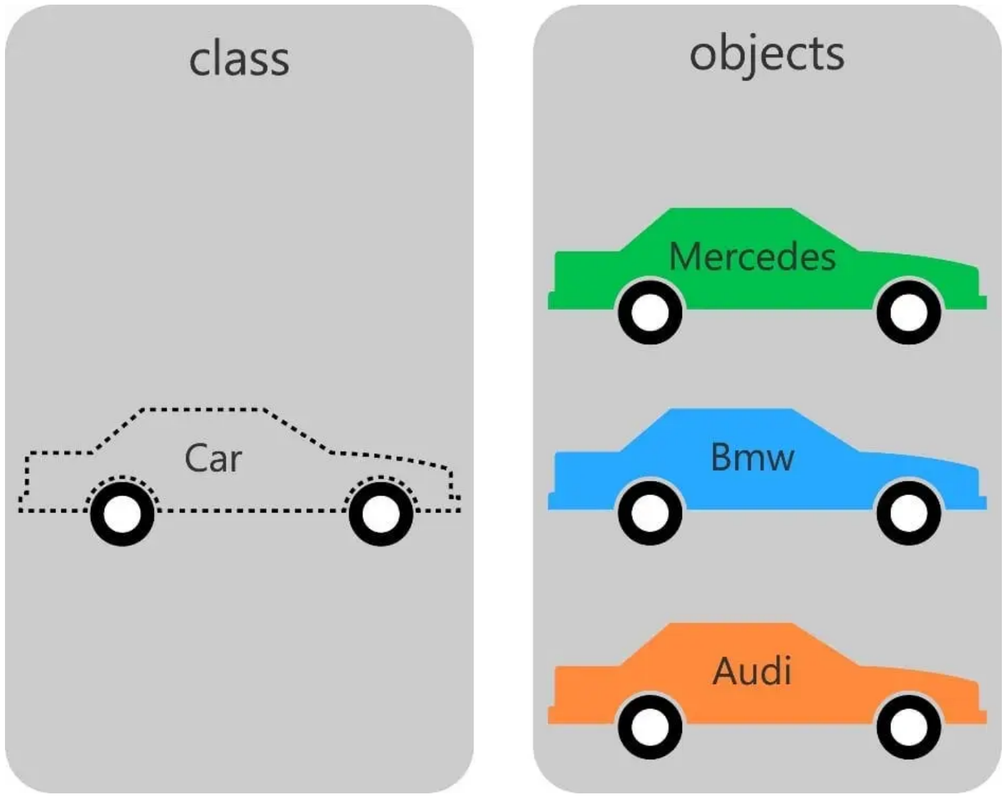
Je interageert met objecten door middel van **methodes**. Een methode (Engels: *method*) lijkt op een functie, maar kan de velden van het bijhorende object zien en aanpassen. Een methode stelt de mogelijke acties van een object voor.

Welke methodes zou je toevoegen aan de klasse `Car`?



OGP: TERMINOLOGIE

Je interageert met objecten door middel van **methodes**. Een methode (Engels: *method*) lijkt op een functie, maar kan de velden van het bijhorende object zien en aanpassen. Een methode stelt de mogelijke acties van een object voor.



Welke methodes zou je toevoegen aan de klasse `Car`?

Antwoord:

- rijden
- stoppen
- bijtanken
- ...

SYNTAX IN PYTHON

```python
class Car:
  def __init__(self, color):
    self.color = color
    self.km_travelled = 0

  def drive(self, nb_km):
    self.km_travelled += nb_km
```

- `class` definieert een klasse
- `def` definieert een methode
- `__init__` is een ingebouwde methode van Python. Het is de **constructor** van een klasse
- `self` is een speciale parameter die het object voorstelt waarmee je werkt op dat moment
- `self.naam_attribuut` stelt een attribuut voor van een object van de klasse
- `self.attribuut = nieuwe_waarde` past de waarde van een attribuut aan

SYNTAX IN PYTHON (2)

```python
# Code van vorige slide die de class Car definieert...

my_car = Car('red')
assert my_car.color == 'red'

my_car.drive(10)
my_car.drive(6)
assert my_car.km_travelled == 16
```

- `Car(params)` creëert een nieuwe instantie (object) van de klasse `Car`
- `obj.methode()` roept `methode()` op op `obj`. In de methode wordt `self` vervangen door `obj`.
- Elke klasse definieert automatisch een nieuw **type** met dezelfde naam. Het type van `my_car` is `Car`

AANDACHTSPUNTEN

OM TE ONTHOUDEN: ALGEMEEN

Let op indentatie bij het schrijven van klassen in Python.

Python laat toe dat je nieuwe attributen definieert buiten `__init__()`. **Doe dit echter nooit.** Het is een slechte gewoonte en leidt tot onleesbare 'spaghetticode'.

VOORGEDEFINIEERDE METHODES

Voorgedefinieerde methodes zitten standaard ingebouwd in elke klasse. In Python zijn er 3 om te onthouden:

- `__init__()`: creëert nieuwe instanties van een klasse
- `__repr__()`: geeft een string terug die een object in codetaal voorstelt
- `__str__()`: geeft een string terug die een object in menselijke taal voorstelt

Elke klasse heeft een standaardimplementatie voor deze methodes, maar heel vaak zal je ze moeten **overschrijven**. De standaardimplementatie van `__repr__()` en `__str__()` geeft het geheugenadres van een object terug.

In veel situaties mogen `__repr__()` en `__str__()` hetzelfde teruggeven. Overschrijf in dat geval enkel `__repr__()`. `__str__()` zal dit automatisch overnemen.

NAAMGEVING VAN METHODES

Namen van methodes volgen dezelfde schrijfregels als voor functies. In Python komt dit neer op `snake_case`.

Programmeurs hebben bepaalde gewoontes bij het kiezen van namen voor methodes:

- `is` of `has` als de methode een `boolean` waarde van een attribuut teruggeeft
- `get` als de methode een niet-booleaanse waarde van attribuut teruggeeft
- `set` als de methode een attribuut aanpast

Deze methodes noemt men ook wel de *getters* en *setters* van een klasse.

Methodes die hier niet onder vallen, hebben meer vrijheid qua naamgeving. In de praktijk wordt ook bij die methodes de woorden `is, has, get, set` veel gebruikt. Ook functies volgen vaak die conventies!

OM TE ONTHOUDEN

`self` is **altijd** de eerste parameter van elke methode. Python dwingt dit niet af, maar zal rare fouten geven tijdens uitvoering als je het vergeet.

Let vanaf nu nóg meer op naamgeving van methodes en functies. Gebruik `is, has, get, set` waar het past om de leesbaarheid te verhogen.

OM TE ONTHOUDEN (2)

Objectgericht programmeren introduceert veel nieuwe termen. Let als leraar op je woordgebruik:

- Functies en methodes zijn strikt genomen niet hetzelfde
- Haal termen zoals klassen, objecten, instanties, attributen, ... niet door elkaar
- Besef ook welke termen wél hetzelfde betekenen (bv. attributen = velden = kenmerken)

STANDAARDWAARDEN IN METHODES EN FUNCTIES

STANDAARDWAARDEN

```python
class Car:
  def __init__(self, color="gray"):
    self.color = color


new_car = Car()
assert new_car.color == "gray"
```

Standaardwaarden (Engels: *default values*) worden automatisch gebruikt als een waarde ontbreekt voor een parameter.

Voordeel: Je moet minder parameters schrijven bij het maken van objecten.

STANDAARDWAARDEN (2)

Standaardwaarden werken ook voor parameters in functies. Het kunnen er ook meer dan 1 zijn:

```python
def get_prediction(temperature_celsius, pressure_bar=1, wind_speed_kmh=10):
    # Code die de weersvoorspelling bepaalt en teruggeeft
```

Aandachtspunten:

- Parameters met standaardwaarden komen altijd **als laatste**. Python dwingt dit af
- Niet elke parameter *moet* een standaardwaarde hebben. Zie het als iets 'extra'

DEFAULT VALUES BIJ OPROEPING

Wanneer je met parameters met default values werkt, kan je naar specifieke parameters verwijzen bij het oproepen. Bijvoorbeeld:

```python
class Car:
  def __init__(self, color="gray", brand="Nissan"):
    self.color = color
    self.brand = brand


new_car = Car(brand="Mercedes", color="blue")
assert new_car.color == "gray"
assert new_car.brand == "Mercedes"
```

Aandachtspunten:

- Syntax is `naam=value`, **zonder spaties**
- Parameters van default values kan je van plaats verwisselen. In het voorbeeld zijn `brand` en `color` omgewisseld wanneer het object wordt gemaakt

KLASSEN COMBINEREN

KLASSEN COMBINEREN

Stel dat we een veerboot (Engels: *Ferry*) willen modelleren. De veerboot kan wagens vervoeren.

Met behulp van klassen kunnen we dit eenvoudig modelleren:

Python

```
class Ferry:
  def __init__(self):
    self.cars_on_board = []

  def onboard_car(self, car):
    self.cars_on_board.append(car)
```

Een `Ferry` is een nieuwe klasse die meerdere `Car`s kan bevatten.

OBJECTEN EN CALL SEMANTICS

Klassen zijn blauwafdrukken voor objecten. Klassen en objecten kunnen redelijk complex worden. Om geheugengebruik te optimaliseren, gebruikt Python (en bijna elke andere taal) standaard **call by reference** voor objecten.

Het gevolg hiervan is dat objecten met elkaar verbonden kunnen worden door referenties naar elkaar op te slaan. Dit kan leiden tot een 'ketting' van data.

Python en vele andere talen zijn achter de schermen zelf opgebouwd als een complex 'netwerk' van klassen.

GEVOLGEN VAN CALL BY REFERENCE

Stel dat we twee veerboten hebben. Op beide veerboten wordt telkens een blauwe wagen geladen. Je zou dit als volgt kunnen programmeren:

```python
ferry_1 = Ferry()
ferry_2 = Ferry()

blue_car = Car('blue')
ferry_1.onboard_car(blue_car)
ferry_2.onboard_car(blue_car)
```

Dit lijkt correct, maar er zit een logische fout in. Welke?

GEVOLGEN VAN CALL BY REFERENCE

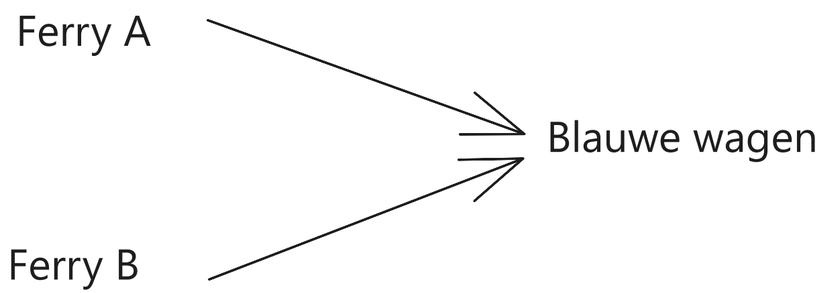
Stel dat we twee veerboten hebben. Op beide veerboten wordt telkens een blauwe wagen geladen. Je zou dit als volgt kunnen programmeren:

```python
ferry_1 = Ferry()
ferry_2 = Ferry()

blue_car = Car('blue')
ferry_1.onboard_car(blue_car)
ferry_2.onboard_car(blue_car)
```

Dit lijkt correct, maar er zit een logische fout in. Welke?

Beide veerboten slaan een referentie op naar dezelfde wagen. Het is alsof de wagen op de twee boten tegelijk aanwezig is, wat in werkelijkheid niet kan.



CALL BY REFERENCE DOORBREKEN

Soms is call by reference niet gewenst wanneer je werkt met objecten. In die gevallen kan je het probleem oplossen door kopieën van objecten te maken.

In Python:

```python
import copy

blue_car = Car('blue')
ferry_1.onboard_car(blue_car)
ferry_2.onboard_car(copy.copy(blue_car))
```

1. Importeer de module `copy` (deel van de Python standard library)
2. Roep `copy(obj)` op uit de module. De methode geeft een kopie terug van `obj`

`copy()` VERSUS `deepcopy()`

Python heeft nog een tweede methode: `deepcopy()`. Ze werkt gelijkaardig aan `copy()`, maar er is een subtiel verschil:

```python
# ferry_a is een oppervlakkige kopie:
# de boot is gekopieerd, maar de wagens op de boot niet
ferry_b = copy.copy(ferry_a)

# in ferry_c zijn zowel de boot als alle
# wagens op de boot kopieën van ferry_a
ferry_c = copy.deepcopy(ferry_c)
```

Welke je nodig hebt, hangt af van de situatie en de code.

> **i** Merk op dat voor objecten van `Car` het resultaat van `copy()` en `deepcopy()` hetzelfde is. Waarom?

De code hieronder test het gedrag van `copy()` en `deepcopy()` op `Ferry`. Welke `assert`(s) zal/zullen **falen**?

```python
ferry_1 = Ferry()
my_car = Car("yellow")
ferry_1.onboard_car(my_car)

ferry_2 = copy.copy(ferry_1)
assert ferry_1 == ferry_2  # 1
assert ferry_1.cars_on_board[0] == ferry_2.cars_on_board[0]  # 2

ferry_3 = copy.deepcopy(ferry_1)
assert ferry_1 != ferry_3  # 3
assert ferry_1.cars_on_board[0] == ferry_3.cars_on_board[0]  # 4
```

- 1 en 4
- 2 en 3
- 1
- 3

GELIJKHEID TUSSEN OBJECTEN

Twee objecten zijn gelijk (==) als hun referentie hetzelfde is. Met andere woorden: ze refereren naar dezelfde plaats in het geheugen.

Een kopie van een object heeft een andere plek in het geheugen en dus ook een andere referentie.

DETAILS VAN `deepcopy()`

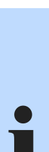
`deepcopy()` gaat automatisch alle subobjecten van een object kopiëren. Dit werkt voor 1 laag, 2 lagen, ... n lagen van objecten.

Welk concept past Python hier achter de schermen toe? Tip: denk terug aan de vorige lessen.

DETAILS VAN `deepcopy()`

`deepcopy()` gaat automatisch alle subobjecten van een object kopiëren. Dit werkt voor 1 laag, 2 lagen, ... n lagen van objecten.

Welk concept past Python hier achter de schermen toe? Tip: denk terug aan de vorige lessen.



`deepcopy()` werkt **recursief**. De base case is dat een object geen subobjecten heeft.

OPDRACHTEN

OPDRACHT 1

Doorloop de theorie in het handboek en maak de extra opgaven rond punten en rechthoeken tussendoor. Maak dan Opgave 20.1 op p.249.

Dit telt samen als één oefening voor het portfolio.

OPDRACHT 2

Versimpelde versie van handboek > p. 249 > Opgave 20.2, zonder leeftijd.

Een student heeft een voornaam, een achternaam, een geboortedatum (bestaande uit jaar, maand, en dag) en een administratienummer. Een cursus heeft een naam en een nummer. Studenten kunnen zich inschrijven voor cursussen. Creëer een class `Student` en een class `Cursus`. Creëer een aantal studenten en een aantal cursussen. Schrijf iedere student in voor een paar cursussen. Toon een lijst van studenten, die hun nummer, voornaam, achternaam en geboortedatum toont, en per student alle cursussen waarvoor de student is ingeschreven.

Je mag extra klassen creëren indien gewenst.

OPDRACHT 3

Breid de code van de wagens en de veerboten uit. Elke wagen heeft nu een gewicht in kg. Elke veerboot heeft een limiet op het aantal kg dat ingeladen mag worden. De standaardwaarde is 100.000kg. Zorg voor een manier om eenvoudig de huidige belasting te berekenen.

Voordat een wagen op een veerboot kan rijden, wordt eerst het aantal aanwezige wagens geteld. Als de boot reeds vol is, gebeurt er niets en loopt het programma gewoon verder.

Schrijf zelf testen met `assert` om te bewijzen dat het aantal wagens op de boot niet toeneemt wanneer de gewichtslimiet overschreden zou worden.

OPDRACHT VAN VORIGE JAREN

In de lerarenopleiding van een Vlaamse hogeschool vinden verkiezingen plaats voor de samenstelling van de opleidingscommissie (OC) en programmaraad (PR). Elke lector kan zich opgeven als kandidaat voor de OC, of de PR, of beide. Stemgerechtigd zijn de studenten: een student mag 1 stem uitbrengen voor de OC en 1 voor de PR. Het moet mogelijk zijn lijsten te tonen met de kandidaten die zich inschreven voor ofwel de OC ofwel de PR. Van elke kandidaat moeten daarin de naam, de leeftijd en het geslacht vermeld worden. Voor een bepaalde verkiezing (OC of PR) moet op aanvraag een lijst kunnen getoond worden met de “stand”: hoeveel stemmen behaalden de kandidaten (al)? Een dergelijke lijst moet gerangschikt zijn in volgorde van afnemende hoeveelheid stemmen, en kunnen getoond worden voor ofwel alle ingeschreven kandidaten, ofwel enkel de mannelijke, ofwel enkel de vrouwelijke. De “stand”-lijst met alle kandidaten moet bovendien ook het totaal aantal (al) uitgebrachte stemmen voor de betreffende commissie of raad aangeven.

Ontwerp, implementeer en test een objectgericht/objectgebaseerd programma in Python dat deze functionaliteiten biedt.