



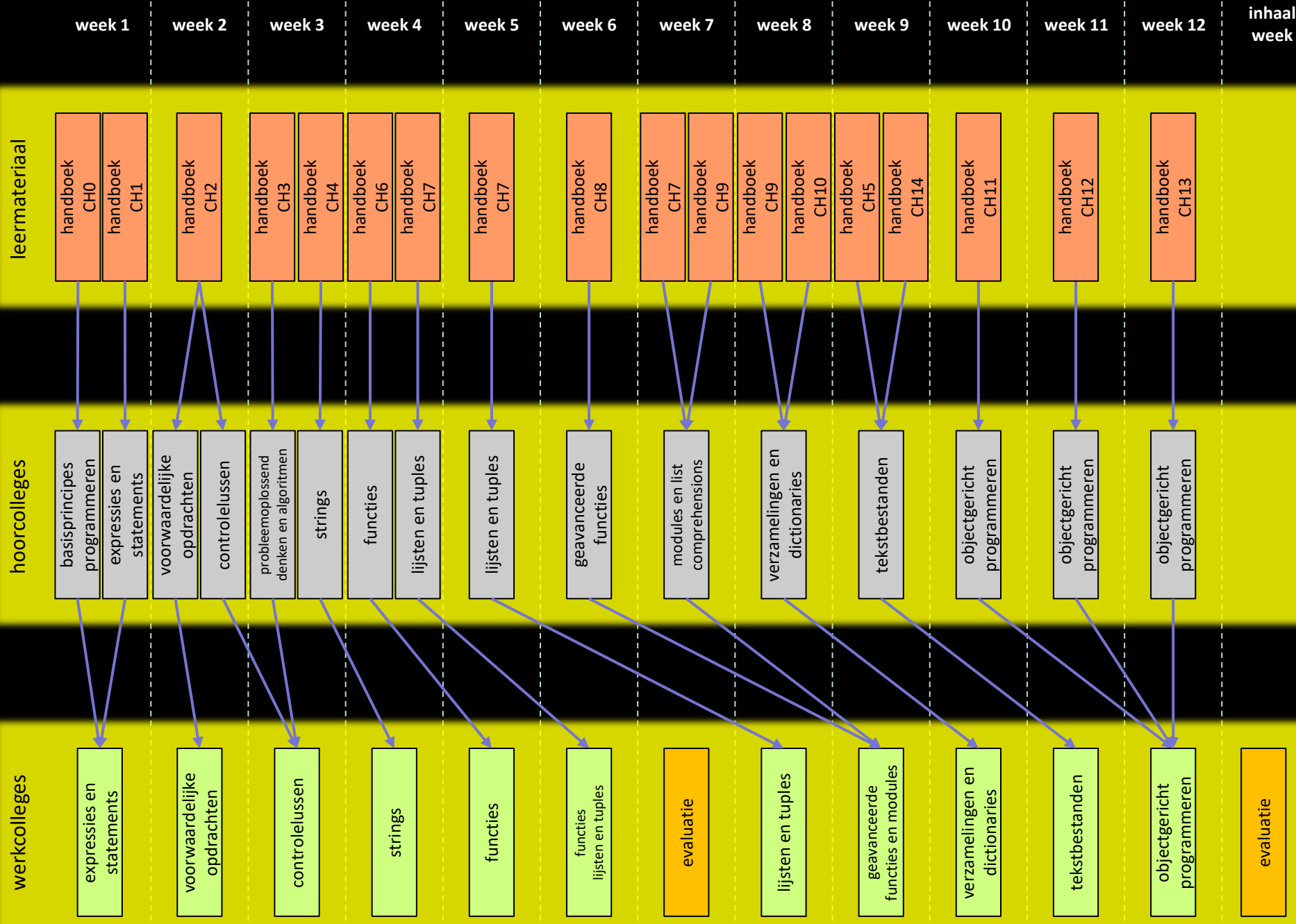
Programmeren in Python

inleiding tot objectgericht programmeren

prof. dr. Peter Dawyndt

✉ peter.dawyndt@ugent.be

🐦 [@dawyndt](https://twitter.com/dawyndt)



Objectgericht programmeren



- Python is een objectgerichte programmeertaal
 - populair programmeerparadigma sinds mid jaren '80
 - gericht op ontwikkelen van grote en complexe software

*aanmaken van objecten
die zowel gegevens als
manipulaties bundelen*

objectgericht
programmeren

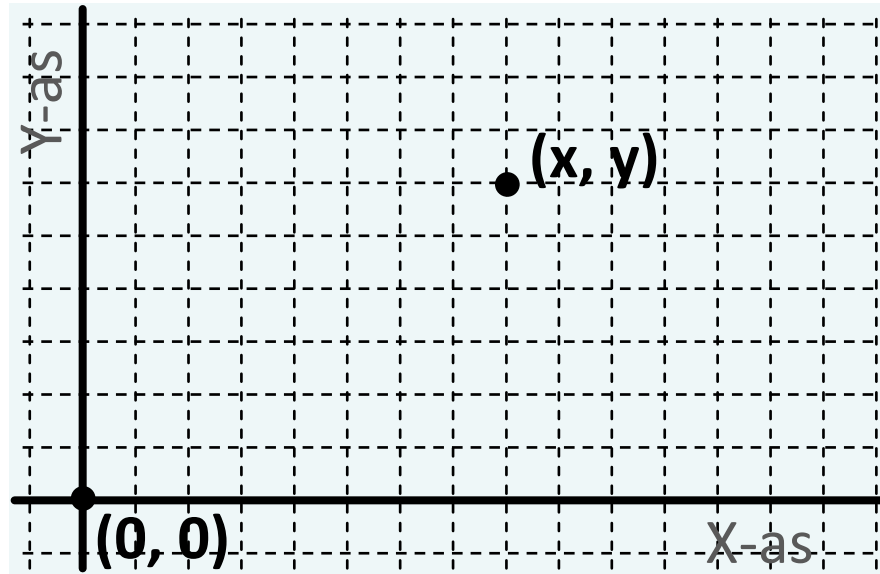
*schrijven van functies die
gegevens manipuleren*

procedureel
programmeren

Zelf gegevenstypes definiëren



- **klasse**: definieert nieuw gegevenstype
 - uitbreiding op ingebouwde gegevenstypes



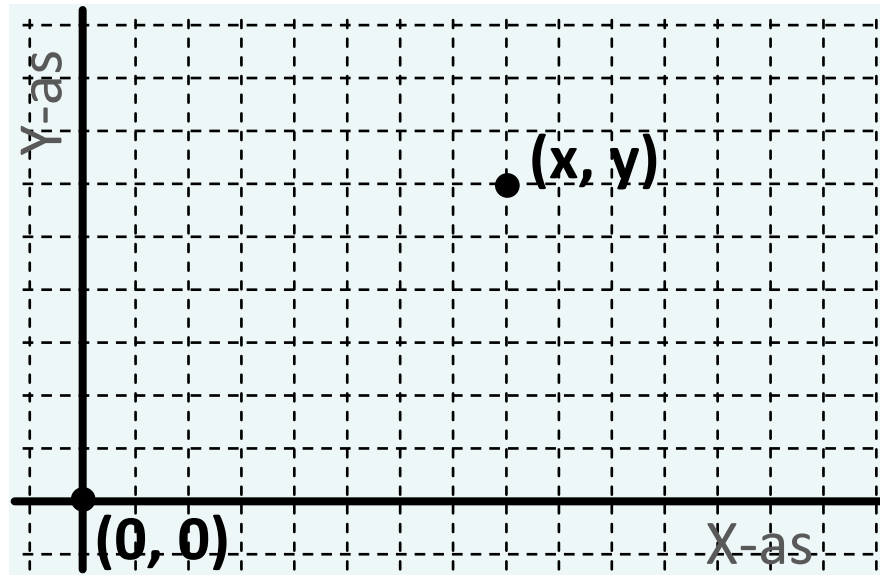
```
class Punt:  
    pass
```



Zelf gegevenstypes definiëren



- **klasse**: definieert nieuw gegevenstype
 - uitbreiding op ingebouwde gegevenstypes



```
class Punt:  
    '''Gevenstype voor 2-dimensionale punten'''
```



Zelf gegevenstypes definiëren

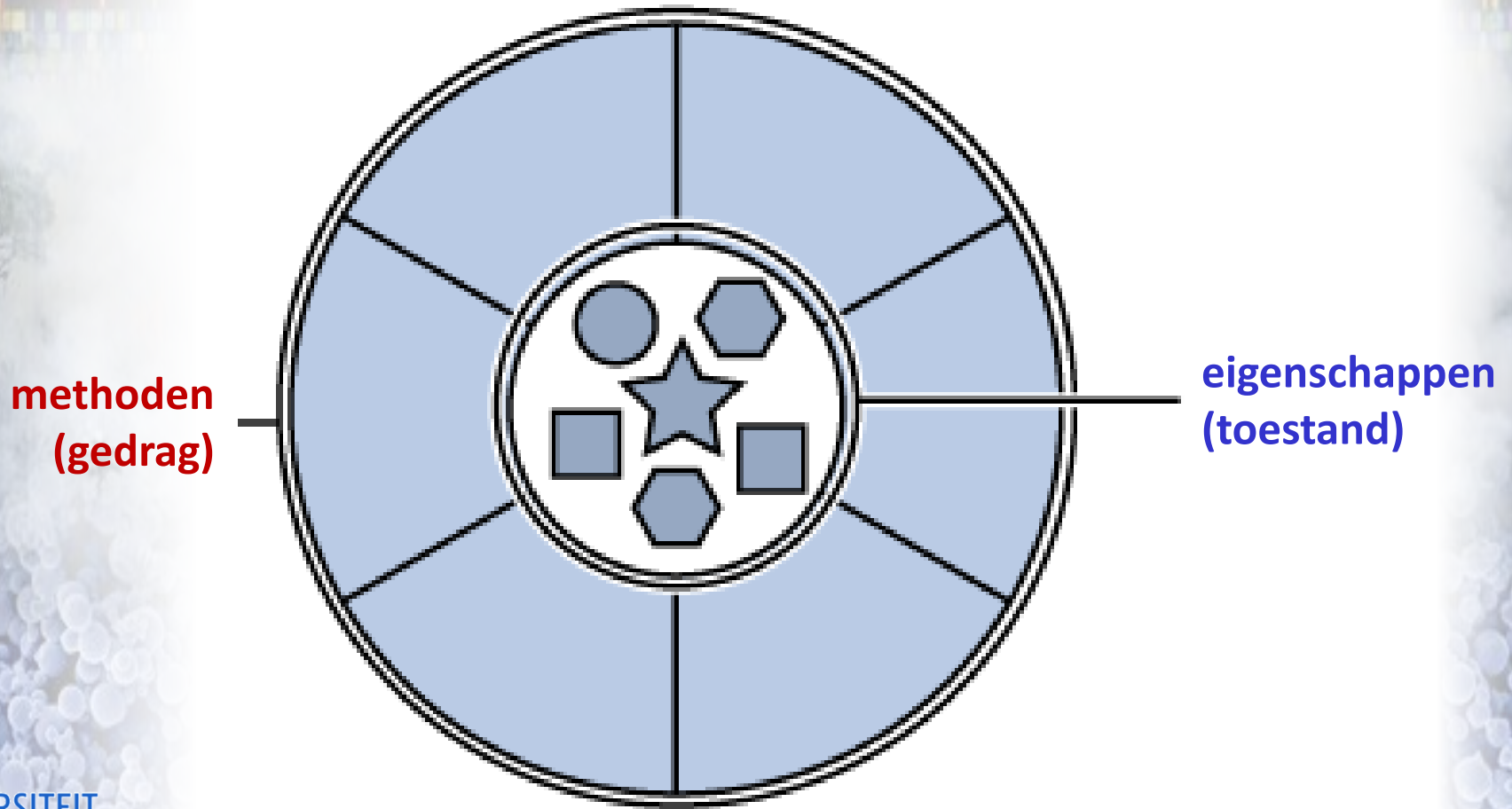


- **klasse**: definieert nieuw gegevenstype
 - Python laat toe om abstracte gegevenstypes te definiëren
 - *abstract data types* (ADT)
 - "abstract" omdat ze implementatiedetails verbergen
 - interactie gebeurt via beperkte set van methoden, eerder dan rechtstreeks gegevens te manipuleren (*encapsulation*)
 - hierdoor kan er (in theorie) minder mislopen
 - programmacode is makkelijker te lezen
 - programmacode is makkelijker te onderhouden

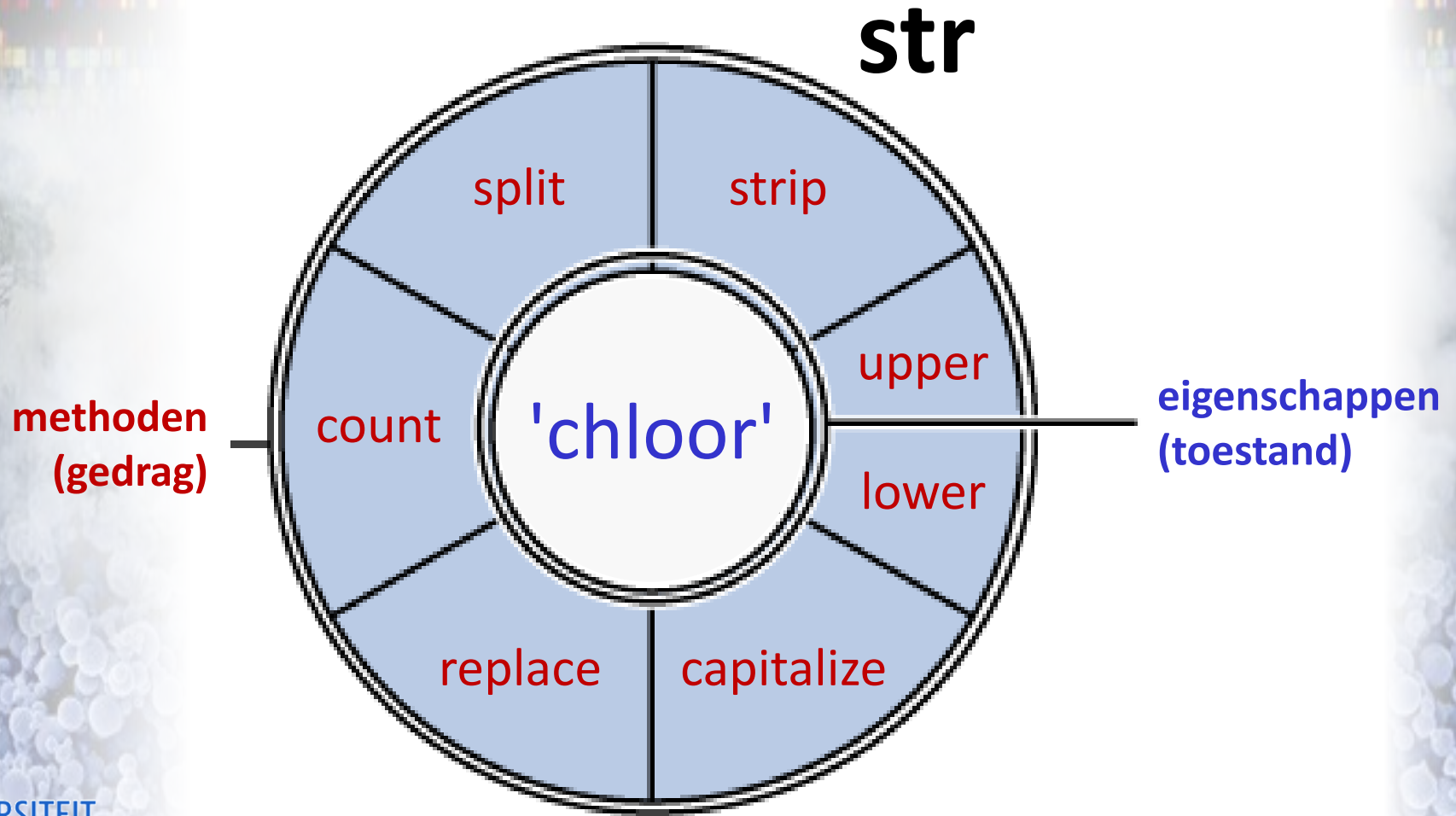
```
class Punt:  
    '''Gevenstype voor 2-dimensionale punten'''
```



Zelf gegevenstypes definiëren



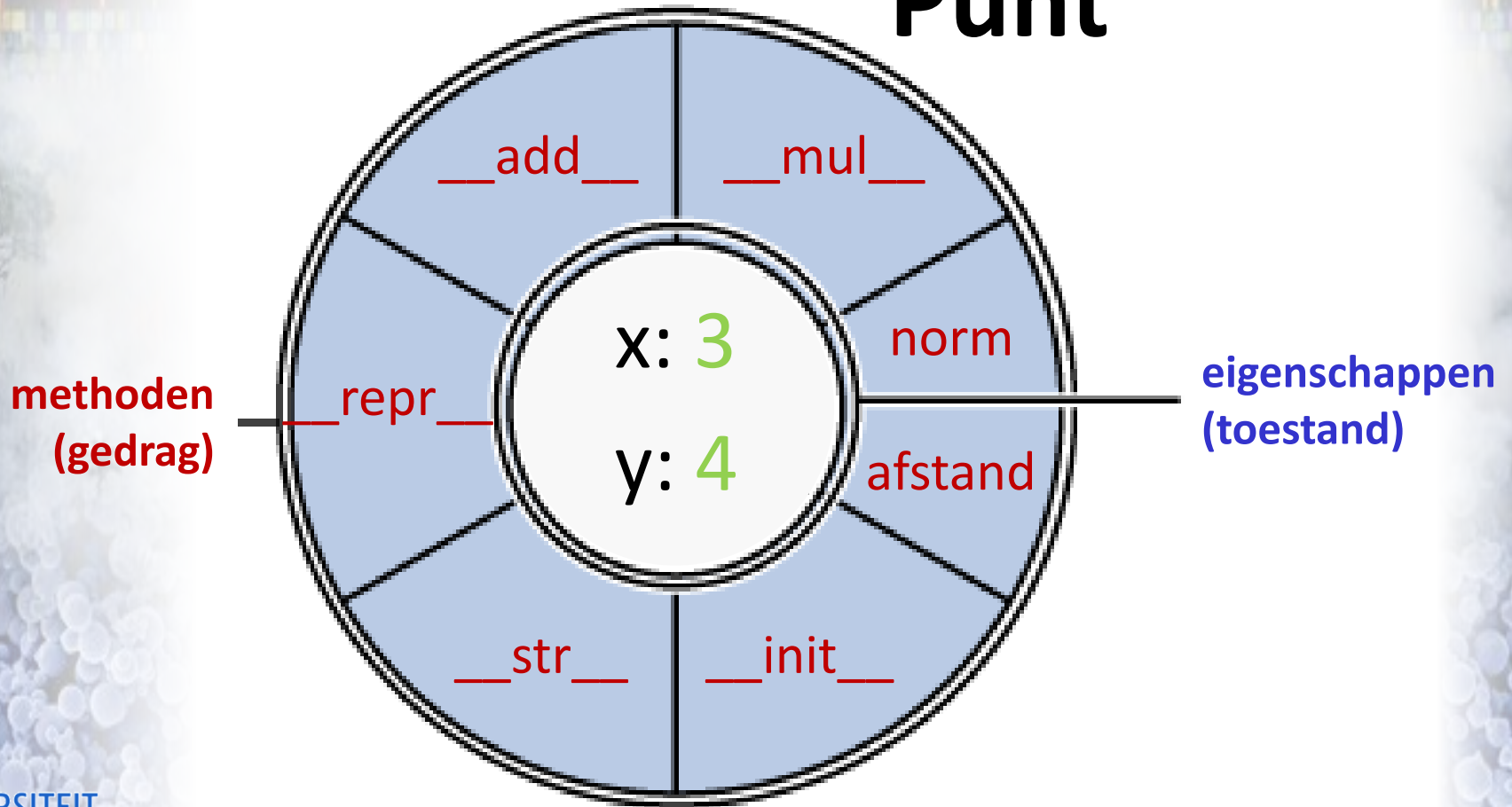
Zelf gegevenstypes definiëren



Zelf gegevenstypes definiëren



Punt



Klassen en instanties



61.8"

57.2"

DEEP-SKIRTED ENGINE BLOCK GIVES ADDED STEEL SUPPORT TO BEARINGS AND CRANKSHAFT.

ALUMINUM PISTONS ARE REINFORCED WITH STEEL STRUTS FOR HEAT-EXPANSION CONTROL.

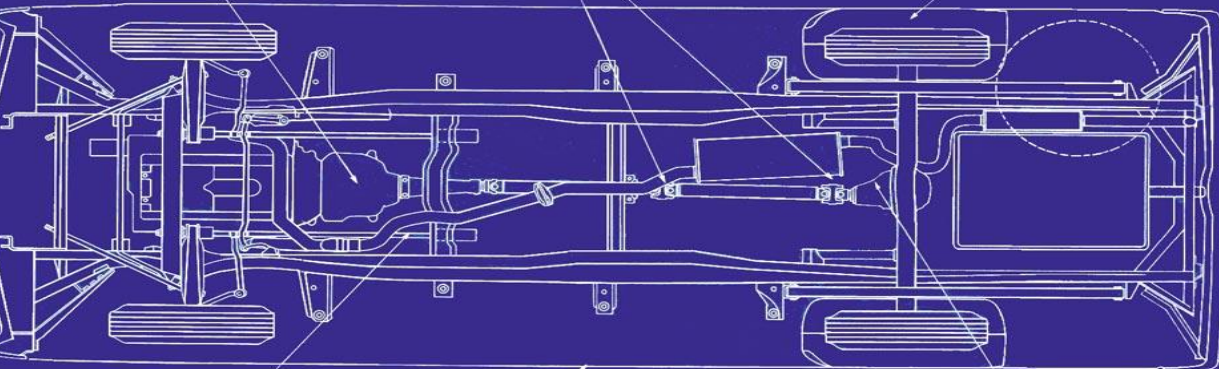
IMPERIAL'S EXECUTIVE, AIRCRAFT TYPE SEATS ARE INDIVIDUALLY POWER-ADJUSTED. THE SEAT ON THE PASSENGER'S SIDE CAN BE TILTED BACK INTO ANY ONE OF FIVE DIFFERENT POSITIONS.

FORGED-STEEL CRANKSHAFT, DYNAMICALLY BALANCED... HAS MICROFINISHED BEARING SURFACES TO REDUCE FRICTION, MINIMIZE WEAR.

IMPERIAL'S 3-SPEED TORQUE-CONVERTER AUTOMATIC TRANSMISSION IS ABOUT THE MOST RESPONSIVE AVAILABLE IN ANY AUTOMOBILE (IT'S A BETTER VERSION OF THE TYPE MANY DRAG-STRIP DRIVERS PREFER.)

TWO-PIECE DRIVESHAFT WITH TWO CONSTANT-VELOCITY JOINTS ACCOMMODATES IMPERIAL'S LONG WHEELBASE... KEEPS DRIVESHAFT RUNNING TRUE, MINIMIZES VIBRATION.

IMPERIAL'S NEW, LOWER PROFILE TIRES OFFER IMPROVED HANDLING, REDUCE TIRE SQUEAL, INSURE LONGER TREAD LIFE.



HIGH CHROME STEEL TORSION BARS SOAK UP ROUGHEST ROAD SHOCKS BEFORE THEY REACH THE CAR BODY.

THE BODY OF AN IMPERIAL UNDERGOES A 15-STEP RUST-PREVENTIVE TREATMENT. SOME OF THE INITIAL SOLUTIONS USED, ACTUALLY INCREASE THE STEEL'S RESISTANCE TO CORROSION, BY CHANGING THE MOLECULAR SURFACE STRUCTURE.

DIFFERENTIAL GEARS ARE EXTRA STRONG TO HANDLE HIGH ENGINE TORQUE... ARE HELICAL CUT TO PROVIDE BETTER GEAR-MATING SURFACES, QUIETER OPERATION.

SPECIFICATIONS

ENGINE: OVERHEAD VALVE 90 DEGREE V-8, 413 CU. IN. DISPLACEMENT, 10.1 TO 1 COMPRESSION RATIO, 340 HP @ 4600 RPM, TORQUE, 470 LB.-FT. @ 2800 RPM.

FUEL SYSTEM: FOUR-BARREL CARBURETOR WITH MECHANICALLY CONTROLLED SECONDARY BARRELS, AUTOMATIC CHOKE, POSITIVE THROTTLE RETURN. FUEL TANK CAPACITY, 23 GALLONS.

ELECTRICAL SYSTEM: 12 VOLT BATTERY, 78 PLATES, 70 AMP.-HR. RATING, 35-AMP. ALTERNATOR (46 AMP. WITH AIR CONDITIONING).

TRANSMISSION: TORQUEFLITE AUTOMATIC WITH COLUMN-MOUNTED SELECTOR LEVER. THREE-SPEED PLANETARY GEAR SET WITH INCREASED HELIX ANGLE. TRANSMISSION BREAK-AWAY RATIO... 4.90 TO 1. IMPROVED TORQUE CONVERTER.

FRAME: FOR CLOSED MODELS... PERIMETER-TYPE LADDER FRAME WITH SIX CROSS-MEMBERS, FULL-LENGTH OUTBOARD SIDE RAILS.

SUSPENSION: CHROME-STEEL TORSION-BAR INDEPENDENT FRONT WHEEL SUSPENSION. BALL-JOINT PIVOTS. HOTCHKISS DRIVE. LEAF-TYPE REAR SPRINGS, 60 IN. LONG, MOUNTED 45 1/2 INCHES APART. ORIFLOW SHOCK ABSORBERS AT ALL FOUR WHEELS. REAR AXLE STABILIZER STRUTS.

STEERING: FULL-TIME POWER STEERING 35 TURNS, FULL LEFT TO FULL RIGHT... SYMMETRICAL IDLER-ARM STEERING LINKAGE. HYDRAULIC AND MECHANICAL STEERING REACTION SYSTEMS. ADJUSTABLE STEERING WHEEL OPTIONAL AT EXTRA COST.

BRAKES: SELF-ADJUSTING POWER BRAKE SYSTEM, FLARED BRAKE DRUMS, BONDED LININGS; TOTAL EFFECTIVE BRAKING AREA, 287 SQ. IN. MECHANICAL PARKING BRAKE WITH AUTOMATIC RELEASE.

WHEELS AND TIRES: LOW PROFILE TYPE 9.15 x 15, TUBELESS TIRES ON SAFETY-RIM WHEELS. STAINLESS STEEL WHEEL COVERS.

DIMENSIONS: FOR CLOSED MODELS... WHEELBASE, 129 IN. FRONT TREAD, 61.5 IN.; REAR, 61.7 IN. OVERALL LENGTH, 227.0 IN. WIDTH, 80.0 IN. HEIGHT (LOADED) 57.2 IN.

SCALE: 1" = 1' 3"

129" WHEELBASE IMPERIAL CROWN COUPE

DRAWING:

Walt Blythe

THE INCOMPARABLE
IMPERIAL
VINTAGE 1965

Klassen en instanties



- ADT wordt gecreëerd door definitie van klasse
 - geeft aan hoe toestand wordt voorgesteld (**eigenschappen**)
 - geeft aan wat instanties van klasse kunnen doen (**methoden**)
 - klassen zijn zelf ook objecten, net als functies
- objecten worden aangemaakt als instanties van klassen
 - alle objecten van bepaalde ADT hebben zelfde methoden, maar individuele waarden voor hun eigenschappen
 - object wijzigen heeft geen invloed op toestand ander object

```
class Punt:
```

```
    '''Gegevenstype voor 2-dimensionale punten'''
```



Klassen en instanties



- klasse **Punt** creëert nieuw gegevenstype: **Punt**
 - **instantiation**: nieuw object aanmaken
 - wordt gedaan door klasse aan te roepen
 - klassen zijn net als functies 'aanroepbaar' (*callable*)
 - leden van gegevenstype: **instanties** of **objecten**

```
>>> class Punt:
...     pass
...
>>> type(Punt)
type
>>> p = Punt()
>>> type(p)
__main__.Punt
```

Klassen en instanties



- bekijk klasse als blauwdruk om objecten te maken
 - klasse **Punt** is blauwdruk om punten te maken
 - klasse **Punt** is zelf geen instantie van een punt, maar de nodige machinerie om punten te kunnen aanmaken



INSTANCE

INSTANCE

INSTANCE

INSTANCE

INSTANCE

```
>>> class Punt:
...     pass
...
>>> type(Punt)
type
>>> p = Punt()
>>> type(p)
__main__.Punt
```

Eigenschappen



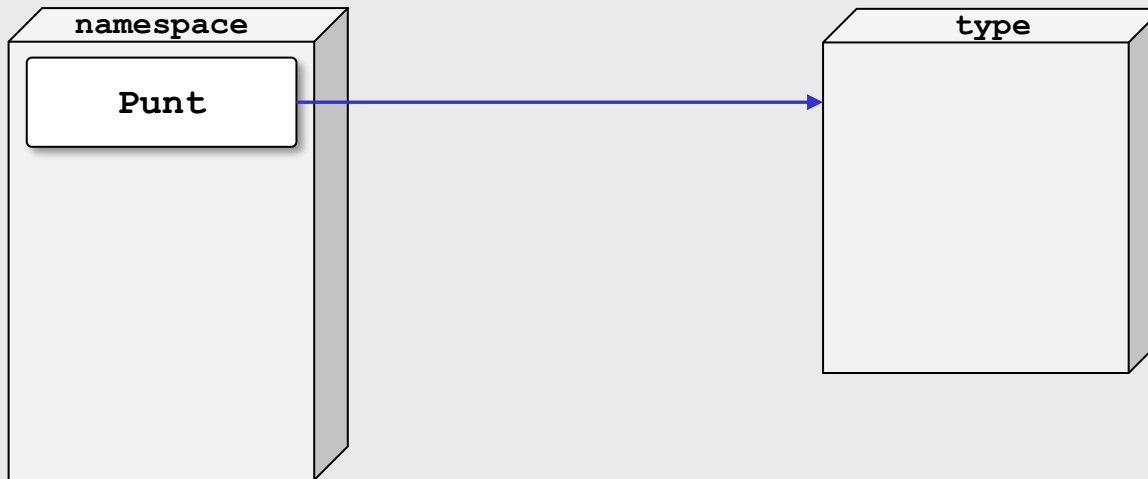
- objecten hebben een toestand en een gedrag
 - toestand bepaald door eigenschappen van instanties
 - instanties vormen eigen *namespace* (net als modules)
 - **attribuut**: naam binnen namespace (instanties en modules)
 - gebruik **dot-notatie** om naam in namespace aan te spreken
 - ook bij modules: **math.pi**, **string.ascii_uppercase**

```
>>> class Punt:
...     pass
...
>>> p = Punt()
>>> p.x = 3
>>> p.y = 4
```

Eigenschappen



objecten (computergeheugen)

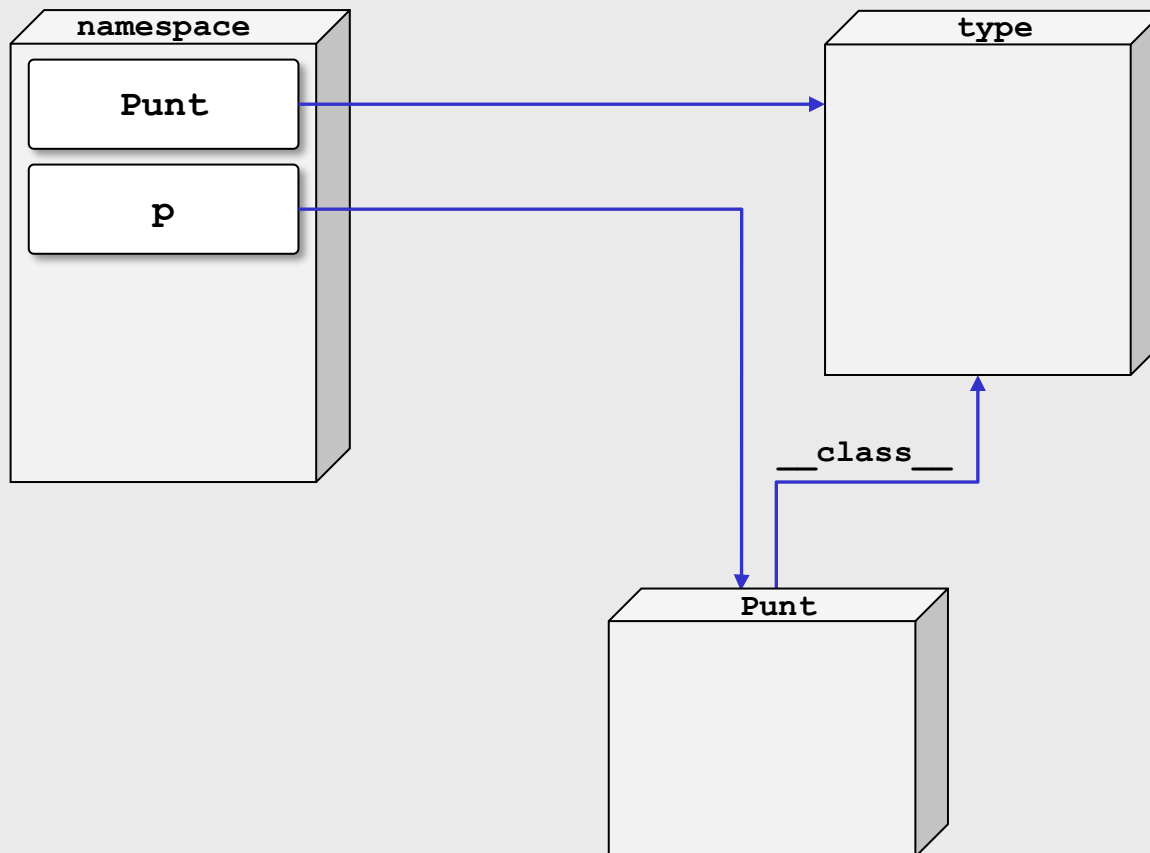


```
>>> class Punt: pass
```

Eigenschappen



objecten (computergeheugen)

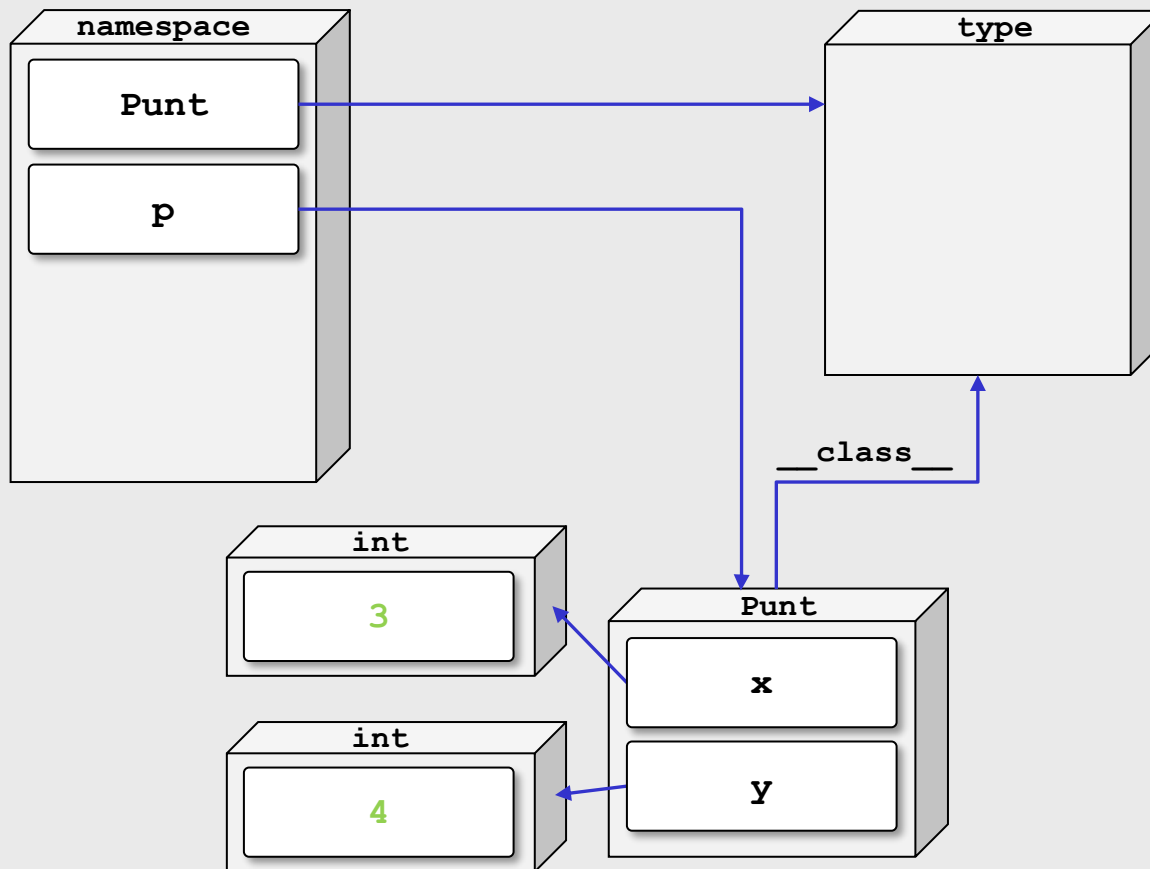


```
>>> p = Punt()
```

Eigenschappen



objecten (computergeheugen)



```
>>> p.x, p.y = 3, 4
```

Eigenschappen



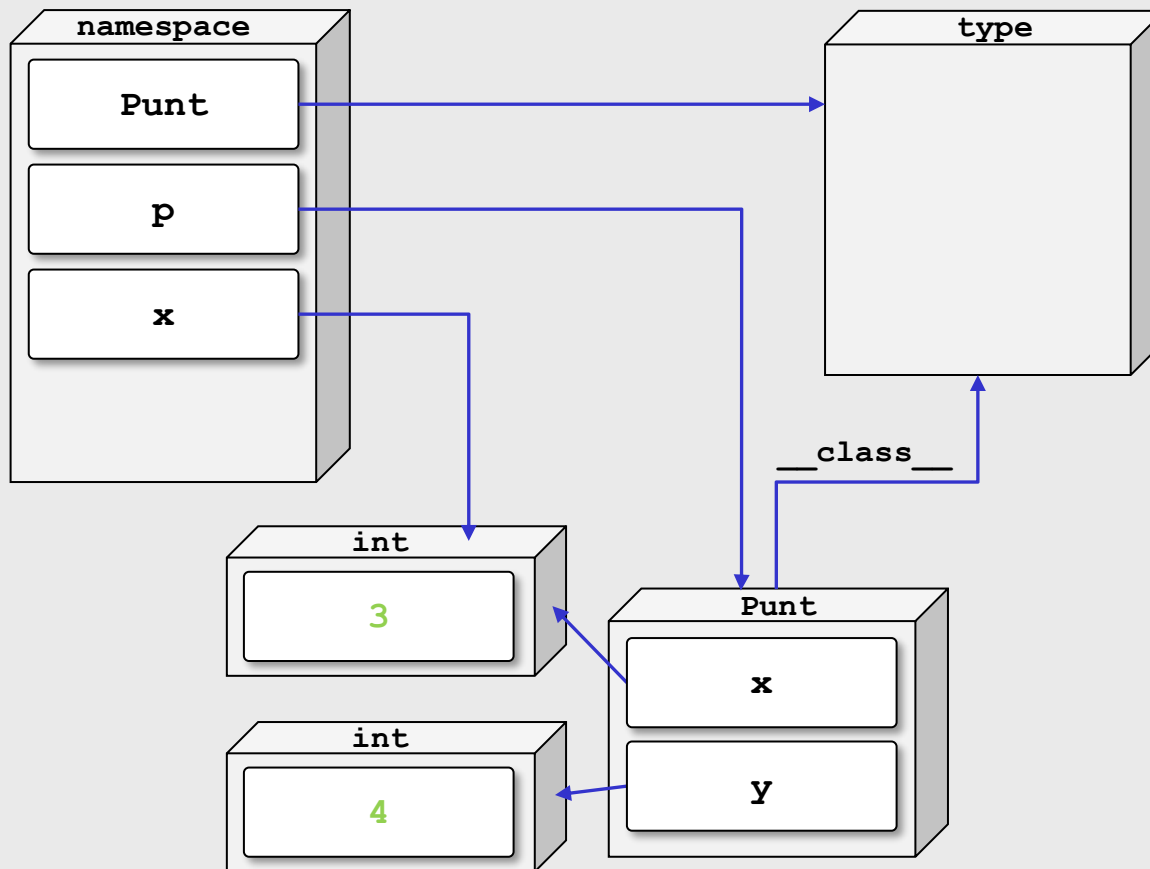
```
>>> class Punt:
...     pass
...
>>> p = Punt()
>>> p.x = 3
>>> p.y = 4
>>> p.y
4
>>> x = p.x
```



Eigenschappen



objecten (computergeheugen)



```
>>> x = p.x
```

Eigenschappen



```
>>> class Punt:
```

```
...     pass
```

```
...
```

```
>>> p = Punt()
```

```
>>> p.x = 3
```

```
>>> p.y = 4
```

```
>>> p.y
```

```
4
```

```
>>> x = p.x
```

```
>>> x
```

```
3
```

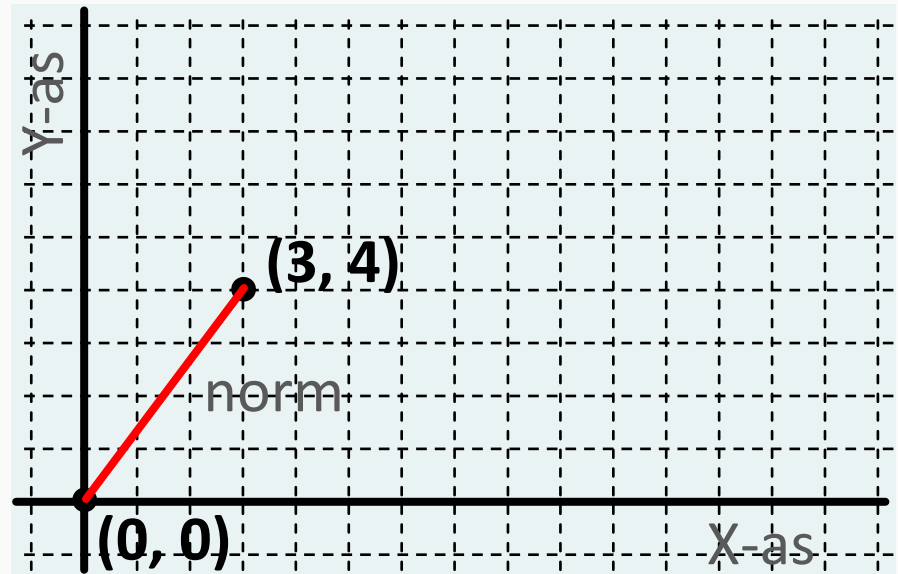
```
>>> print(f'({p.x:.2f}, {p.y:.2f})')
```

```
(3.00, 4.00)
```

```
>>> norm = (p.x ** 2 + p.y ** 2) ** 0.5
```

```
>>> norm
```

```
5.0
```



Methoden



- voeg methode toe door functie te definiëren in klasse
 - instantie altijd als eerste argument doorgegeven aan methode
 - conventie: geef eerste parameter naam **self**
 - in tegenstelling tot **this** in C++ en Java is deze naam een conventie
 - iedereen gebruikt die, dus wijk daar best niet van af

```
>>> class Punt:
...     def norm(self):
...         return (self.x**2 + self.y**2) ** 0.5
...
>>> p = Punt()
```

Methoden



- voeg methode toe door functie te definiëren in klasse
 - methode aanroepen: `object.methode(argumenten)`
 - dit zoekt de klasse `K` waarvan `object` een instantie is
 - roept daarna `K.methode(object, argumenten)` aan

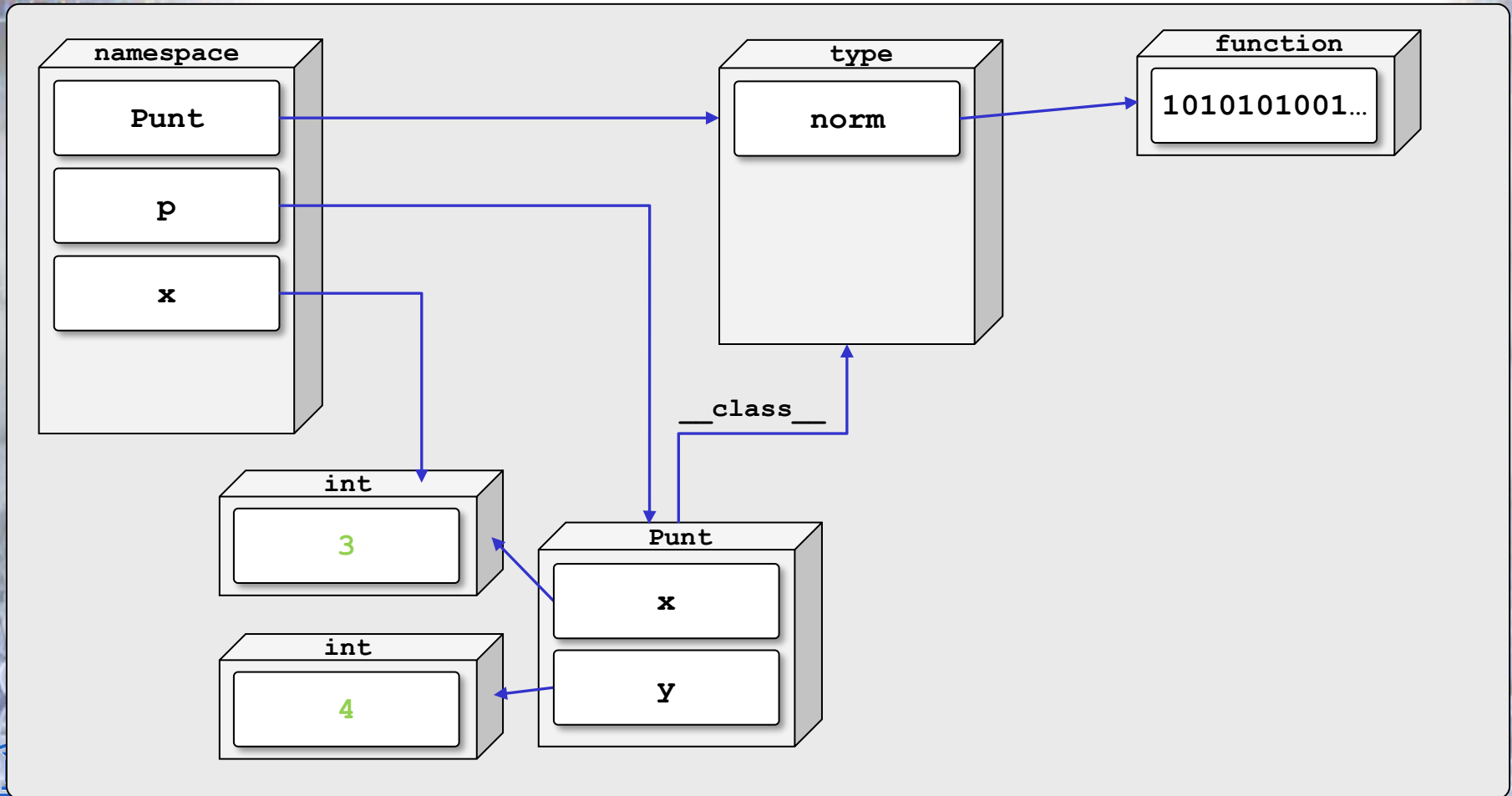
```
>>> class Punt:
...     def norm(self):
...         return (self.x**2 + self.y**2) ** 0.5
...
>>> p = Punt()
>>> p.x, p.y = 3, 4
>>> p.norm()
5.0
>>> Punt.norm(p)
5.0
```



Methoden



objecten (computergeheugen)

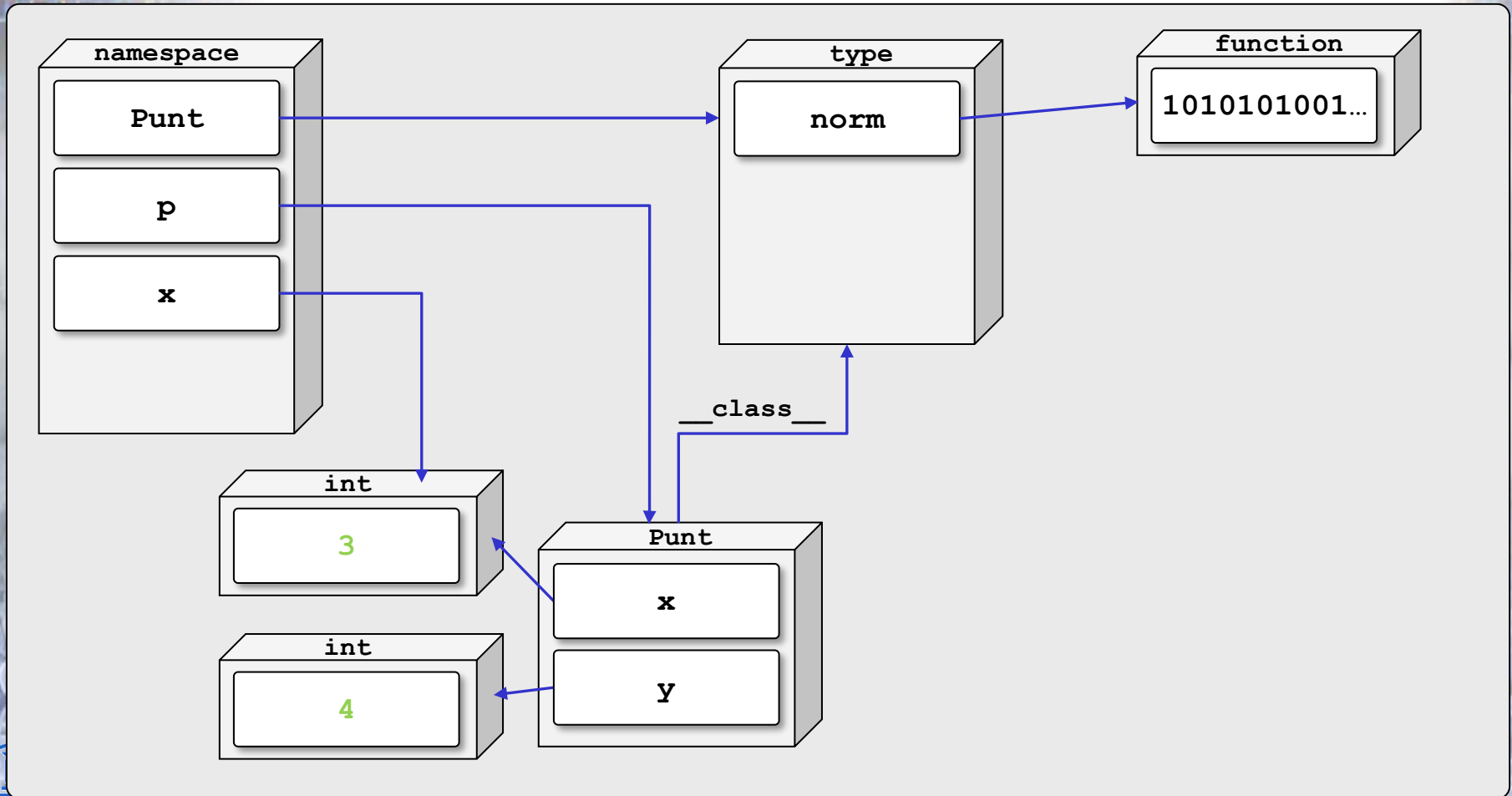


```
>>> p.norm()
```

Methoden



objecten (computergeheugen)



```
>>> Punt.norm(p)
```

Initialisatiemethode



- klasse **Punt** stelt 2-dimensionale punten voor
 - *alle* punten moeten attributen **x** en **y** hebben

```
>>> q = Punt()
```

```
>>> q.x
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
AttributeError: Punt instance has no attribute 'x'
```



Initialisatiemethode



- klasse **Punt** stelt 2-dimensionale punten voor
 - *alle* punten moeten attributen **x** en **y** hebben
 - initialisatiemethode **__init__** initialiseert **Punt** objecten
 - automatisch aangeroepen als object aangemaakt wordt
 - wordt daarom ook vaak *constructor* genoemd
 - analoog: **list()** maakt een lijst, **str()** maakt een string

punt1.py

```
class Punt:
```

```
    def __init__(self, x=0, y=0):  
        self.x = x  
        self.y = y
```

```
    def norm(self):  
        return (self.x ** 2 + self.y ** 2) ** 0.5
```



Initialisatiemethode



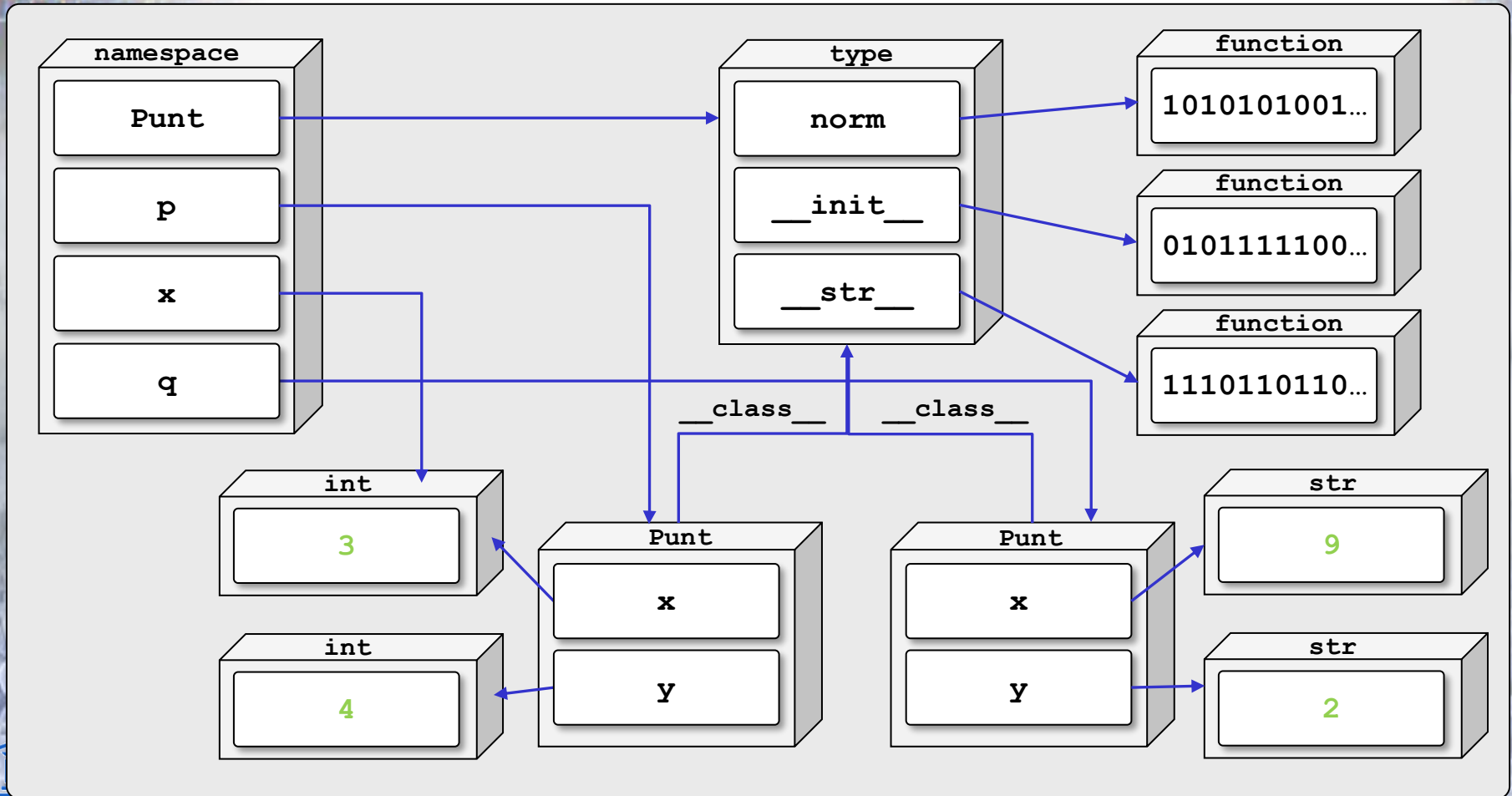
```
>>> from punt1 import Punt
>>> p = Punt(3, 4)
>>> p.x
3
>>> p.y
4
>>> p.norm()
5.0
>>> x = p.x
>>> q = Punt(9, 2)
>>> q.x
9
>>> q.y
2
>>> q.norm()
9.2195444572928871
```



Initialisatiemethode



objecten (computergeheugen)



```
>>> p, q = Punt(3, 4), Punt(9, 2)
```

Instanties als parameters

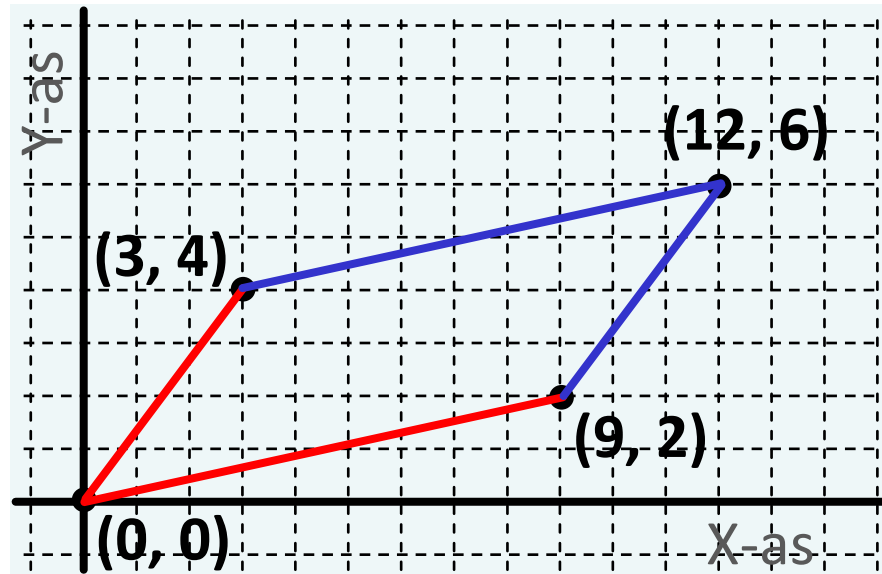


- objecten kunnen op de gebruikelijke manier als argument aan functies doorgegeven worden

```
>>> p, q = Punt(3, 4), Punt(9, 2)
>>> def toon(p):
...     print(f'({p.x:.2f}, {p.y:.2f})')
>>> toon(p)
(3.00, 4.00)
>>> def vectorsom(p1, p2):
...     return Punt(p1.x + p2.x, p1.y + p2.y)
```



Instanties als parameters



```
>>> def vectorsom(p1, p2):  
...     return Punt(p1.x + p2.x, p1.y + p2.y)
```



Instanties als parameters



- objecten kunnen op de gebruikelijke manier als parameter aan functies doorgegeven worden

```
>>> p, q = Punt(3, 4), Punt(9, 2)
>>> def toon(p):
...     print(f'({p.x}, {p.y})')
>>> toon(p)
(3.00, 4.00)
>>> def vectorsom(p1, p2):
...     return Punt(p1.x + p2.x, p1.y + p2.y)
>>> r = vectorsom(p, q)
>>> isinstance(r, Punt)
True
>>> toon(r)
(12.00, 6.00)
```

Operator overloading



Operator overloading



- methoden gedragen zich als functies, maar
 - methoden worden gedefinieerd binnen klassedefinitie
 - expliciete relatie tussen klasse en methode
 - syntaxis om methode aan te roepen verschilt van syntaxis om functie aan te roepen

punt1.py

```
class Punt:

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def norm(self):
        return (self.x ** 2 + self.y ** 2) ** 0.5
```



Operator overloading



```
>>> from punt2 import Punt
>>> p, q = Punt(3, 4), Punt(9, 2)
>>> p.toon()
' (3.00, 4.00) '
>>> r = p.vectorsom(q)
>>> r.toon()
' (12.00, 6.00) '
```

punt2.py

```
class Punt:

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def norm(self):
        return (self.x ** 2 + self.y ** 2) ** 0.5

    def toon(self):
        return f'({self.x:.2f}, {self.y:.2f})'

    def vectorsom(self, other):
        return Punt(self.x + other.x, self.y + other.y)
```



Operator overloading



- ingebouwde functies en operatoren toegepast op `o` automatisch omgezet naar ingebouwde methoden van `K`
 - `str(o) → K.__str__(o)`
 - geeft voorstelling van object als string terug
 - functie `str` impliciet aangeroepen door functie `print`
 - `repr(o) → K.__repr__(o)`
 - interne voorstelling van object als string (initialisatiemethode)
 - `len(o) → K.__len__(o)`
 - `o + o2 → K.__add__(o, o2)`
 - `o * o2 → K.__mul__(o, o2)`

```
class K: pass  
o = K()
```



Operator overloading



```
>>> from punt3 import Punt
>>> p, q = Punt(3, 4), Punt(9, 2)
>>> p
Punt(3, 4)
>>> r = p + q
>>> print(r)
(12.00, 6.00)
```

punt3.py

```
class Punt:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
    def norm(self):
        return (self.x ** 2 + self.y ** 2) ** 0.5
    def __str__(self):
        return f'({self.x:.2f}, {self.y:.2f})'
    def __repr__(self):
        return f'Punt({self.x}, {self.y})'
    def __add__(self, other):
        return Punt(self.x + other.x, self.y + other.y)
```



Romeins rekenen



I = 1
II = 2
III = 3
IV = 4
V = 5
VI = 6
VII = 7
VIII = 8
IX = 9
X = 10

XL = 40
L = 50
LX = 60
LXX = 70
LXXX = 80
XC = 90
C = 100
D = 500
M = 1000

I VIEW X-RAYS



LUCKY COWS
DRINK MILK



Klassen: 'new style' vs 'old style'



- Python 2.x
 - onderscheid tussen 'new style' en 'old style' klassen
 - verschil zit enkel in geavanceerde (technische) details

```
class Punt:  
    pass
```

*old
style*

```
class Punt(object):  
    pass
```

*new
style*

- Python 3.x
 - geen onderscheid tussen 'new style' en 'old style' klassen

```
class Punt:  
    pass
```

*new
style*

```
class Punt(object):  
    pass
```

*new
style*

OOP karakteristieken



E N C A P S U L A T I O N

P O L Y M O R P H I S M

I N H E R I T A N C E

Huiswerk (volgende les)



- handboek
 - lees hoofdstuk 12 (*more on classes*)
 - lees hoofdstuk 13 (*OOP development*)
 - let in het bijzonder op gebruikte OOP terminologie
- voorbeelden volgende les (reeks 10)
 - immuunsysteem
 - dominostenen
 - romeins rekenen



Huiswerk (werkcolleges)



- *tweede evaluatie*
 - *maandag 16 december 2024 (14:30-16:45)*
 - *dinsdag 17 december 2024 (10:00-12:15)*
 - *inschrijven in Ufora groepen*

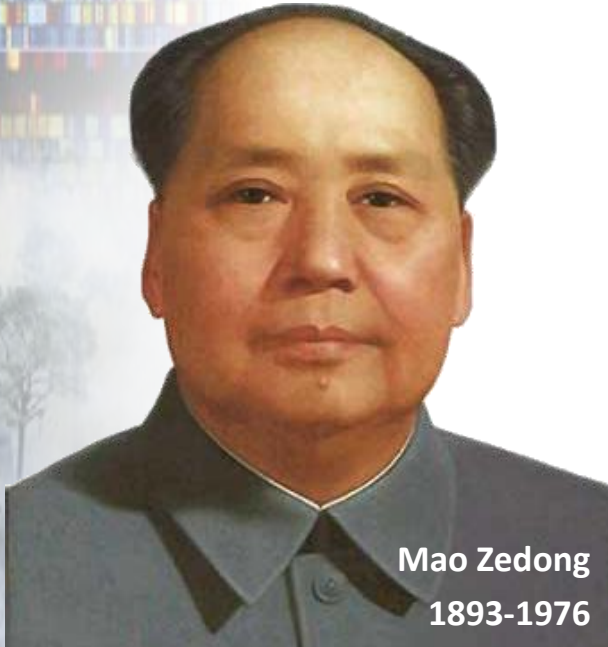


Vragen of opmerkingen?



UNIVERSITEIT
GENT

The sky is the limit ...



Mao Zedong
1893-1976

"Classes struggle,
some classes triumph,
others are eliminated."

— Mao Zedong



UNIVERSITEIT
GENT