**Starten met classes en objects!**

The four principles of object-oriented programming are **encapsulation**, **abstraction**, **inheritance**,and **polymorphism**.

In python is alles een object, zelfs een integer en een list. oef alles is een object

**Object-oriented programmeren** (OOP) is een gestructureerde manier van programmeren, die bij elkaar horende gegevensstructuren en procedures bundelt in classes.

Met classes kan je een programma opsplitsen in logische onderdelen.

Classes zijn objecten waarmee u gegevensstructuren en procedures op één plek kunt groeperen.

Gebaseerd op deze classes kan men daarna 1 of meerdere objecten maken.

Een class is zelf ook een object en laat toe om een nieuw object te maken dat gebaseerd is op de class. Een class is een soort ontwerp (blueprint) voor de aanmaak van latere objecten.

Een class zal bijvoorbeeld definiëren welke variabelen elk object zal krijgen, en op welke methodes het object zal kunnen beroep doen om zijn object-variabelen te bewerken.

De objecten gemaakt volgens de class gebruiken normaal gezien enkel de gegevensstructuur vastgelegd in de class, en zullen nu ook waardes toekennen aan de variabelen van die gegevensstructuur.

De objecten kunnen de methodes die in de class gedefinieerd worden, gebruiken om hun variabelen te bewerken.

**Samengevat:**

Een class ontwerpt een nieuw data type (gegevensstructuur en methodes)

Een instantie van een class geeft dan een object dat de gegevensstructuur kan vullen met waardes.

Een object bevat variabelen en waardes, geordend volgens de gegevensstructuur gedefinieerd in de class.

Een object kan de functies die in de class gedefinieerd zijn gebruiken op zijn eigen data.

De variabelen + waardes worden opgeslagen in elk individueel object, de functies NIET!

**Misverstanden :**

Classes kunnen geen data bevatten, ze zijn enkel het ontwerp voor de objecten (blueprint)!

Elke object bevat alle gegevens en functies!

Een object is 100% gelijk aan het ontwerp van de class en kan niet meer of minder attributen bevatten.

oef 1 : Maak een class met de naam Test.

Gebruik pythontutor <http://www.pythontutor.com/visualize.html#mode=edit> , dir() en id() om te checken wat er precies gebeurd.

Merk op dat de class wel degelijk bestaat en een plaats krijgt in het geheugen.

Merk ook op dat er veel dunder ingebouwde functies beschikbaar zijn. ( dunder = double underscore)

oef 2 : Maak een class met de naam Test en voeg een class-variabele met de naam teller toe aan de class, geef deze de waarde 100

Bekijk nu in de python-tutor het resultaat !

De Class Test is nu gekend in onze globale namespace.

Test dit met het printen van dir().

Wat zit er precies in de class Test? Vind je de variabele teller terug met dir(Test)?

oef 3: Maak een instantie ( een object) gebaseerd op de class Test uit oef 2, noem het t1.

Kan het object t1 de class-variabele teller lezen?

Zit teller nu ook in het object t1? ( gebruik <http://www.pythontutor.com/>)

check ook met \_\_dict\_\_

Indien niet, waarom lukt t1.teller dan?

oef 4:Verander in oef 3 de waarde van teller naar 200 met t1.teller =200

Wat merk je en hoe verklaar je dit?

Vergelijk dit met wat er in een functie gebeurd zonder het keyword local te gebruiken.

oef 5: Class Functie

Maak een class Test en voeg de functie test\_say\_hello() toe.

Merk je dat Pycharm self toevoegt? Verwijder self.

Maak een object t1 gebaseerd opcdeze class.

Roep deze functie test\_say\_hello() aan vanuit de global scope, de class en het object?

Wat lukt er? Hoe verklaar je dit?

oef 6: object geeft automatisch zijn id naar de functies!

Gebruik en print de default parameter van de functie test\_say\_hello(p=None) voor 2 objecten t1 en t2 gebaseerd op de class Test. Kijk goed in de pythontutor wat er allemaal gebeurt!

Merk op dat er geen variabelen en functies opgeslagen worden in het object.

Merk op dat de functies gedefinieerd in de de class, worden gebruikt door het object, ze worden gerund in het geheugen en maken gebruik van de id’s als parameter . Daarna worden de functies verwijderd uit het geheugen.

Omdat objecten die functies aanroepen, altijd hun id als eerste paramameter meegeven, is het een conventie om die eerste parameter self te noem.

oef 7 : Roep een functie met parameter self aan vanuit de class en het object.

Zoals je merkt geeft Test. test\_say\_hello() geen parameter mee. Om deze reden maken we later een onderscheid tussen :

instance methods

class methods

static methods

oef 8 instance attributes toevoegen nadat het object reeds is gecreëerd.

We hebben dit eigenlijk reeds gedaan ( t1 = 200 )

voeg een attribuut ( variabele) name en lastname toe aan het object t1 en geef deze de waarde name=uwvoornaam lastname=uwachternaam.

Check t1 met dir en \_\_dict\_\_

Besluit :

Het blijkt mogelijk om zomaar attributen toe te voegen aan objecten.

Dit is iets dat in andere talen niet altijd kan/mag!

Ondanks dat het kan in Python zullen we dit VERMIJDEN!

Onze objecten zouden zo te verschillend worden van elkaar, dat is niet de bedoeling.

oef 9 De dunder-method \_\_init\_

Hoe kunnen we object attributen best genereren zodat ze gelijk zijn voor elk object?

Dit gebeurd best tijdens het maken van het object ( instantiëren)

Test de methode \_\_init\_\_

Heeft deze de parameter self nodig?

Het meegeven van self laat alle instance- methodes ( niet enkel deze dunder init) toe alle variabelen in het object terug te vinden!

Dit is zeer belangrijk…

oef10 Geef gewicht, lengte, leeftijd mee tijdens de creatie van een object vh type Persoon.

Check met pythontutor en dir en \_\_dict\_\_

Het is een conventie om dezelfde namen te gebruiken maar dit is niet noodzakelijk.

Moet self 4 keer gebruikt worden? Waarom?

oef 11 Sla ook tijdens de creatie meteen de bmi op voor elke persoon.

Merk op dat niet alle parameters moeten meegeven worden tijdens de creatie om toch objecten te verkrijgen met dezelfde attributen.

oef12 voeg een instance method toe die rapporteert hoe het met je BMI is gesteld.( leeftijd 19-69)

|  |  |
| --- | --- |
| lager dan 18,5 | Ondergewicht |
| Vanaf 18,5 tot 25 | Gezond gewicht |
| Vanaf 25 tot 30 | Overgewicht |
| 30 en hoger | Ernstig overgewicht (obesitas) |

oef 13 : Programmeurs die je class gebruiken vragen nu je class aan te passen zodat er gecheckt kan worden of de ingegeven lengte wel in centimeters is nadat sommige gebruikers de lengte ingaven in meters.

We kunnen dit checken tijdens de creatie maar wat als iemand later een ander gewicht of lengte wil ingeven?

Hoe kunnen we vermijden dat de lengte in meters wordt ingegeven?

**Oplossing > we gebruiken getter en setter methodes!**

Voordeel makkelijk

Nadeel:

1/ men kan alsnog object.lengte gebruiken en foute waarde ingeven > oplossing protected variable gebruiken oef 14

2/ gebruikers van onze class moeten hun implementatie aanpassen. > oplossing property class gebruiken

ipv wim.gewicht = 100 wordt het nu wim.set\_gewicht(100)

oef14 Bewaar de data in protected instance-variabelen

protected => \_naam

private => \_\_naam

dunder method => \_\_naam\_\_

wat is => naam\_ ? 😊

We vertrekken van een class Boss die een instance variabele salaris heeft.

object = Boss()

object.salaris = 50000 enz werkt goed gedurende vele jaren todat iemand = 500000 ingaf..

|  |  |
| --- | --- |
|  |  |

Maak een object met naam diederik\_steinway van het type Boss!

Zorg ervoor dat zijn salaris enkel kan opgegeven en opgevraagd worden met set\_salaris en get\_salaris en dat het onder de 100.000 EUR / jaar blijft. Het salaris wordt opgeslagen in de protected variabele \_salaris!

Opmerking : \_salaris is protected, maar dit is niet meer dan een conventie die zegt dat andere programmeurs deze variabele niet mogen gebruiken. Indien men echt wil lukt dit echter nog steeds!

Besluit :

Dit werkt prima en beperkt nu de waarde die men aan instance-variabelen wil toekennen.

**Het probleem is indien men dit in het begin vergeten was, en een class in omloop is die object.salaris = ondersteunt, men darrna niet wil dat alle software die deze class gebruikt moet aangepast worden!**

oef15 property class gebruiken om salaris aan te passen, te beperken en op te vragen

De grote vraag is, hoe kunnen de andere programmeurs onze class en object.salaris blijven gebruiken zonder hun software aan te passen maar toch de beveiliging er nadien bij krijgen?

**oplossing** : De property class

Deze property class zorgt ervoor dat naar de buitenwerled toe **salaris** een gewoon data- attribuut lijkt, terwijl dit helemaal niet het geval is en bij object.salaris= value of varx= object.salaris in werkelijkheid functies worden opgeroepen.

**syntax :**

naamattribuut – property( get\_attribuut, set\_attribuut) # ook delete mogelijk als 3-de functie

Besluit: de gebruiker heeft geen idee dat intern alles anders werkt dan oorspronkelijk het geval was!

Zijn programma hoeft nu enkel de gewijzigde class te gebruiken om salaris te limiteren.

oef16 property class gebruiken met decorators

De pythonic-way gebruikt decorators, zo kan men beter zien wat de bedoeling is een vermijd men namespace polution. ( meer functies dan nodig?)

# a decorator is simply a function which wraps another function and gives it some addition functionality or modifies the functionality

oef 17-21 Hoe werkt een decorator?

Hier komt heel wat kijken, bestudeer de oefeningen 17-21, uiteindelijk reultaat =

@delay\_decorator

def say\_greeting():

print("How are you?")

# say\_greeting = delay\_decorator(say\_greeting) #is wat er eigenlijk gebeurt ...

oef 22 inheritance of overerving

Bedoeling is om een class te herbruiken en er stukken aan toe te voegen of sommige te wijzigen indien nodig.

Voorbeeld een class Persoon kan reeds vele eigenschappen bevatten die een class Leerling of Leraar ook zou moeten hebben.

We gaan daarom in de class Leraar en Leerling de eigenschappen die gemeenschappelijk zijn herbruiken.

oef23 method overriding

Indien de child class dezelfde methode van de parent class ook gebruikt dan krijgt de methode in de child class voorrang en “override” ze deze in de parent class

oef24 init niet altijd nodig in child class

Indien er geen init wordt gevonden in de child class dan zoekt Python naar boven toe in de hiërarchie tot ze deze methode vindt, vindt ze deze niet dan komt er een error.

oef25 wanneer init child class init super class moet aanroepen

Om het DRY principe te volgen is het best om geen code te herhalen, toewijzingen die reeds in de parent class gebeurden herhalen we liever niet in de child class.

Voorbeeld Persoon, Leraar, Leerling, we sturen met de super().\_\_init\_\_ methode de attributen naar de parent die daar aan het object reeds werden toegekend.

Merk op dat de class Persoon actief bijhoudt hoeveel personen er reeds zijn gecreëerd.

Bij de methode super().\_\_init\_\_ moet men self niet gebruiken , zonder gebruik te maken van de methode super zoals in Persoon.\_\_init\_\_(self,..) wel.

oef26 dunder init methode \_\_str\_\_

Printen van je eigen object geeft het ID van het object.

Je kan beslissen wat er in de plaats wordt geprint met de dunder str methode

Waar de dunder str voor de gebruiker is bedoeld is er een dunder repr die voor de ontwikkelaar is bedoeld.

opmerking : test het voorbeeld met en zonder dunder str

oef27 static en class methods

wanneer geen access tot de object attributen nodig is.

oef 28 argv en kwargs variabel aantal positionele of keyword argumenten

Soms is het noodzakelijk om een felxibel aantal parameters toe te laten.

We hebben nu de keuze uit:

1 parameter

2 parameter met default waarde

3 flexibel aantal positionele parameters met \*

4 flexibel aantal keywoord parameters met \*\*

oef29 composition

Composition betekent dat we classes gebruiken als parameters voor andere classes.

Composition wordt vaker gebruikt dan Inheritance.

oef30 method chaining

Door in een instance method self te returnen kan men bijzondere dingen verkrijgen zoals method chaining.

oef31 hoe call je een method in de superclass vanuit een method of de child class

In de functie van de child class is de functie van de parent niet gekend, enkel de parent class

Check of binnen in de functie display\_area Triangle en /of find\_area gekend zijn

oef32 Device class

Maak een object my\_device gebaseerd op de class Device.

Buiten de Device class gebruik je nog een Led en een Button class.

Ontwerp de Device class zodat je :

led objecten kan toevoegen ( heeft attributen > naam en pin nummer)

button objecten kan toevoegen ( heeft attributen > naam en pin nummer)

Met 3 verschillende methodes :

Kan opvragen welke led objecten jouw device heeft.

Kan opvragen welke button objecten jouw device heeft.

Kan opvragen welke led+button objecten jouw device heeft.

oef33 analyseer wat hier gebeurt in tkinter voorbeeld 1

Welke init wordt er uitgevoerd? Wordt toplevel window Tk gemaakt?