Classes

Wat is een Class?

- Een class is een **type definitie**
- Wij kunnen zelf class typen definieren in python
- Binnen python zijn er ook zeer veel classes voor ons voorgedefinieerd.
- Een class kan dus doorgegeven en in een variabele gestopt worden
- **Een class is een blauwdruk voor een object**
- Een class bevat velden van een bepaald type (kan elk type zijn, ook een andere class)
- Een class bevat ook functies, functies binnen een class worden methods genoemd.

Nieuwe class aanmaken

- Hiervoor gebruiken we de keyword **class**
- Bijv.:

```
class mijnClass:
        eenIntProperty = 5
        eenTextProp = “Text property van mijnClass”
```

Een object uit een class creeren

- Uit een class kan weer een object gecreerd worden
- Bijv.:
  mijnObject = mijnClass()
  print(mijnObject.eenTextProp)

Ingebouwde __init__() functie

- Alle python classes hebben een ingebouwde __init__() functie die automatisch wordt uitgevoerd wanneer het object gecreerd wordt.
- Deze functie kunnen benutten om allerlei (Property) waarden te initieren of andere nodige zaken uit te voeren.
- In andere programmeer talen wordt dit ook vaak de “create” functie genoemd en kan er ook een ingebouwde destroy functie zijn.

Voorbeeld van een __init__() functie

- class Person:
  def __init__(self, name, age):
  self.name = name
  self.age = age
- Self is een keyword variabele die verwijst naar het object waarin het voorkomt (meest programmeertalen hebben dit)
- Is een verwarrend voorbeeld waarin de parameter namen dezelfde zijn als de property namen
- Object wordt als volgt gecreeerd:
  objPersn = Person(“Jan” , 36)

Object methods

- class Person:

```
    def __init__(self, name, age):
    self.name = name
    self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)
```

- Methods zijn dus functies die tot een object behoren (en in de class gedefinieerd zijn).

De self parameter

- De self parameter refereerd naar de object-instantie (dus het gecreerde object) waarin het zich bevind
- Het is altijd de eerste parameter in de __init()__ functie, zoals je daar noemd zo heet die.

Class (object) properties

- De properties van een object kun je wijzigen zoals je een variabel wijzigd.
- Je kan een property van een object verwijderen met keyword del

Een object beeindigen

- Een object wordt beeindigd met keyword del
- Dus bijv:
  del objPersn (objPersn is hier een uit een class gecreerd object)

Waarom moet een object worden gecreerd uit een class?

- Een computer programma bevind zich in twee verschillende soorten werkgeheugen.
- Het ene soort geheugen wordt “the stack” (de stapel) genoemd, de andere “the heap” (de hoop)
- De stack is, als ik het goed zeg, registergeheugen van de processor, die sneller is maar waarvan er minder beschikbaar is als het geheugen van de heap.
- De heap is het werkgeheugen van de computer die in overvloed aanwezig is.
- “Gewone variabelen” (strings, integers, booleans, enz) bevinden zich in de stack en hoeven daarom niet gecreerd te worden.
- Variabelen die “constructies” zijn, zoals objecten en bijv ook array’s, die dus veel groter zijn en meer geheugen gebruiken, kunnen niet op de stack en worden daarom in de heap geplaatst.
- Om een object variabele in de heap te plaatsten zijn allerlei toeters en bellen nodig, (pointers, tellers, buffers enz.) daarom spreakt men van het creeren van een object uit een class.
- Een object variabele in een programma is een pointer, een verwijzing naar een geheugenlocatie in het werkgeheudgen, dus een heel ander soort variable dan bijvoorbeeld een interger of een boolean.

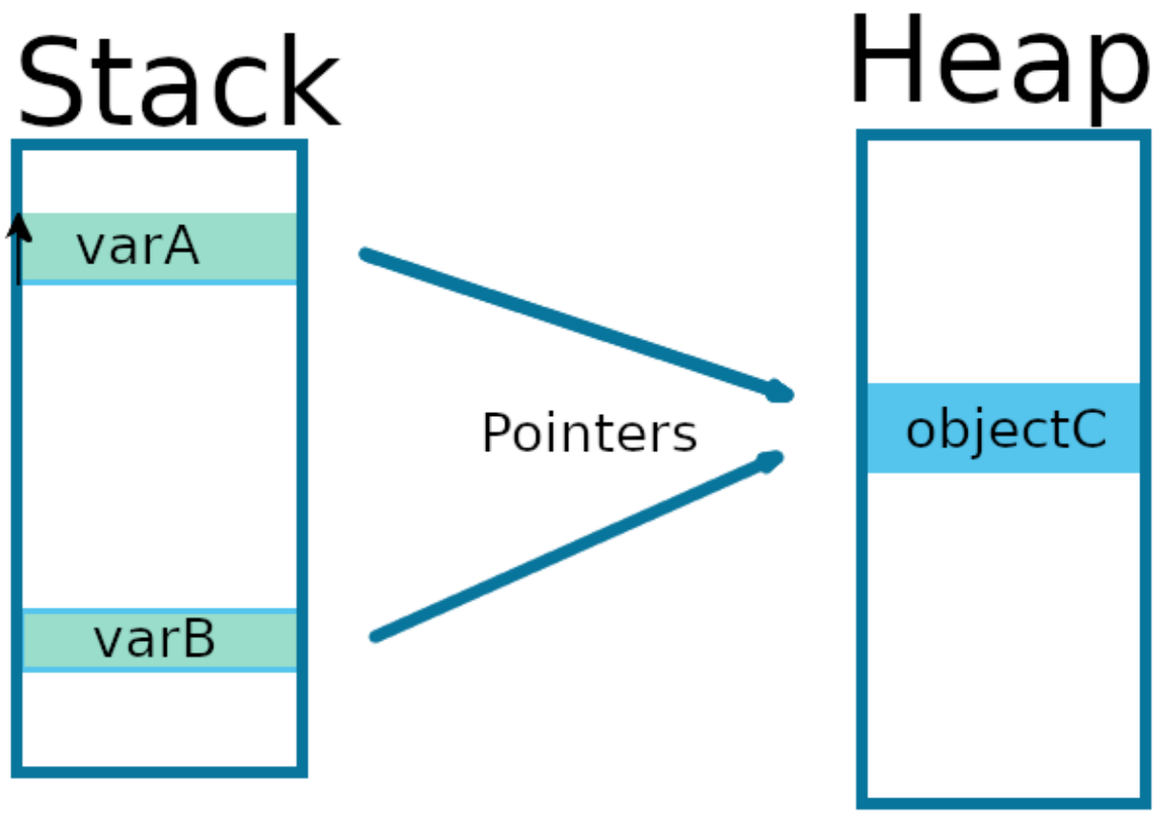
Verschil pointer en gewone variabele

- Wanneer ik een integer variabel in een andere interger varabel stop, dus:
  varA = 5
  varB = varA
  Dan zijn varA en varB aparte copien met getal 5
  Als ik:
  VarB += 1
  doe, dan bevind zich in varA het getal 5 en in varB het getal 6

Verschil pointer en gewone variabele

- Wanneer ik een object variabele aanmaak, en ik stop dit object in een tweede variabele, dus:
  varA = mijnClass()
  varB = varA
  Dan zijn varA en varB **geen** aparte copien elkaar!!!
  Als ik:
  VarB.eenIntProperty = 1001
  doe, dan is ook in varA.eenIntProperty 1001!!!
- Dit komt omdat ik in statement varB = varA, een pointer, dus een verwijzing heb doorgegeven, een verwijzing naar een constructie (een object dus) in het werkgeheugen, varA en varB verwijzen bijde naar hetzelfde object en zijn geen aparte copien zoals bij integers en booleans het geval is.

Stack en heap



Pointers en strings

- Het voorgaande pointer verhaal geld ook voor array’s.
- In veel programmeertalen is een string ook stiekem een array (een array van charackters). In dat geval moet je ook uitkijken als je een stringvariabele doorgeeft aan een andere variabele en hem dan wijzigd!

Classes overerving in Python

Overerving (inheritance)

- Door overerving kunnen we een nieuwe class definieren die gebaseerd is op een andere class en daarmee alle de ouder class zijn properties en methods overerft
- Parent class; de class waarvan wordt georven
- Child class: de (nieuwe) class die alles overerft.

Voorbeeld overerving

```
class Person:
   def __init__(self, fname, lname):
      self.firstname = fname
      self.lastname = lname

  def printname(self):
     print(self.firstname, self.lastname)
```

Nieuwe class (Student)
gebaseerd op class Person

```
class Student(Person):
     pass
```

(gebruik de pass keyword
als je niets wilt toevoegen
aan de class)

Object gecreerd uit
class Student,
dit object erft oa de
method printname over

```
objStudnt = Student("Mike", "Olsen")
objStudnt.printname()
```

Init functie bij child class

- De init functie wordt automatisch aangeroepen wanneer er een nieuw object gecreerd wordt uit een class
- Wanneer in een child class geen init functie is gedefinieerd, dan wordt die van de parent gebruikt.
- Word er in de child class wel een init functie gedef. Dan word die van de parent genegeerd (dit heet “**overriding**”.
- Indien nodig kan vanuit de init van de child de init van de parent worden aangeroepen. (zie voorb volgende dia)

Voorbeeld child class

```
class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
     print("Welcome", self.firstname, self.lastname, \
     "to the class of", self.graduationyear)
```

Override van de parent init, met nieuwe init met extra parameter

Aanroep van de parent init

Nieuwe property (graduationyear)

Nieuwe method (welcome(self))

Tenslotte

- Sommige programmeer talen waaronder waarsch ook python bestaan uit een hele boomstructuur van classes die van elkaar overerven.
- De classes die aan het begin van die boom staan noemen we de base classes.
- Je hebt visuele en non-visuele classes, de meeste zijn non visueel, de visuele classes hebben een zgn canvas property (dit is ook weer een class) waarop getekend kan worden, bijv. buttons, checkboxen enz.
- Bij object georiënteerd programmeren gebruiken we ook veel classes (vaak alleen maar).