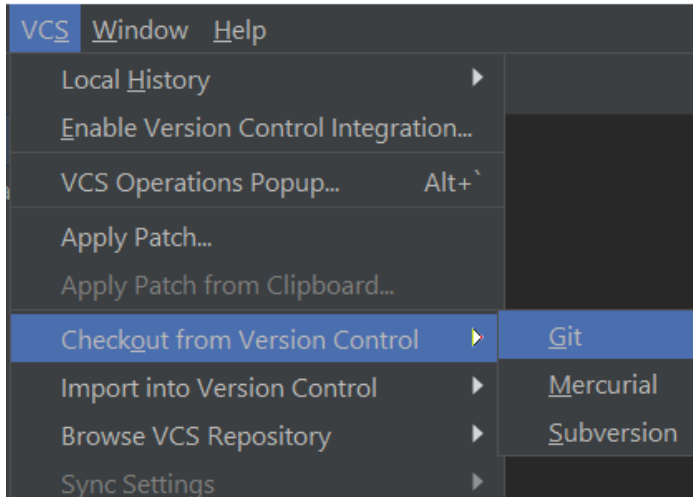


Python Crash Course

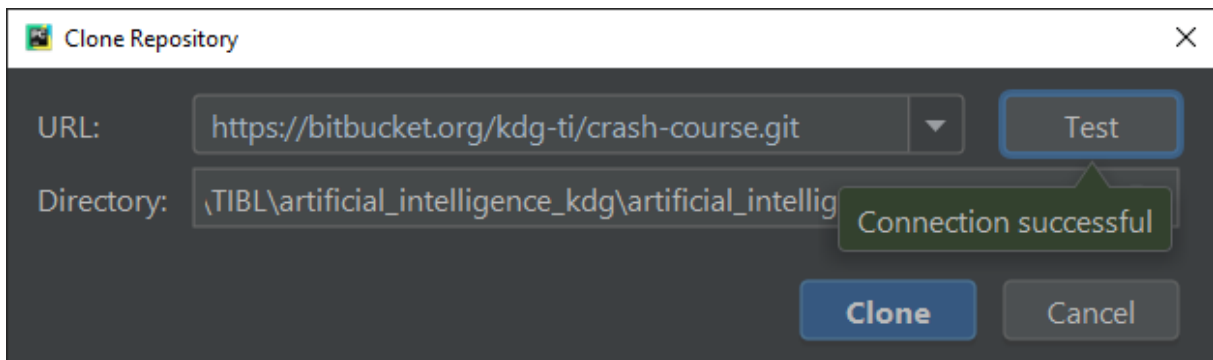
Stap 1: Git repository binnenhalen:

Via menu “VCS > Checkout from Version Control > Git”:

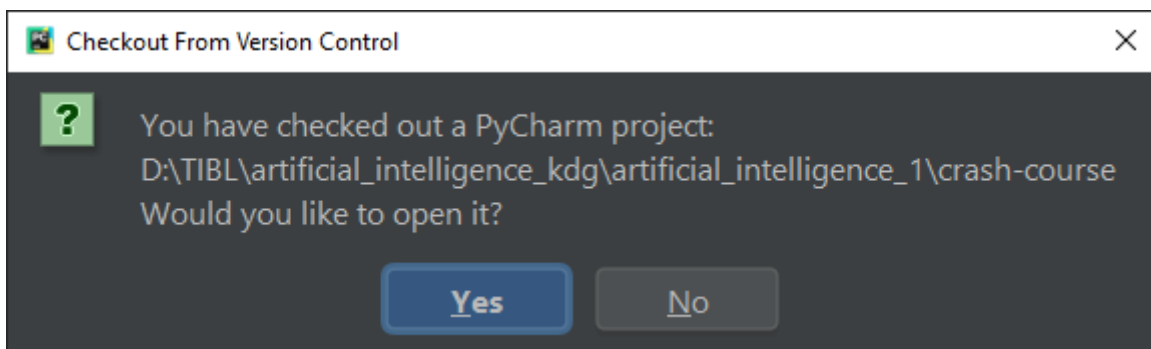
ih_course] - PyCharm



URL naar de GIT repo invullen, eventueel connectie testen en de directory aanpassen, en op “Clone” klikken:



Dan op “Yes” klikken:

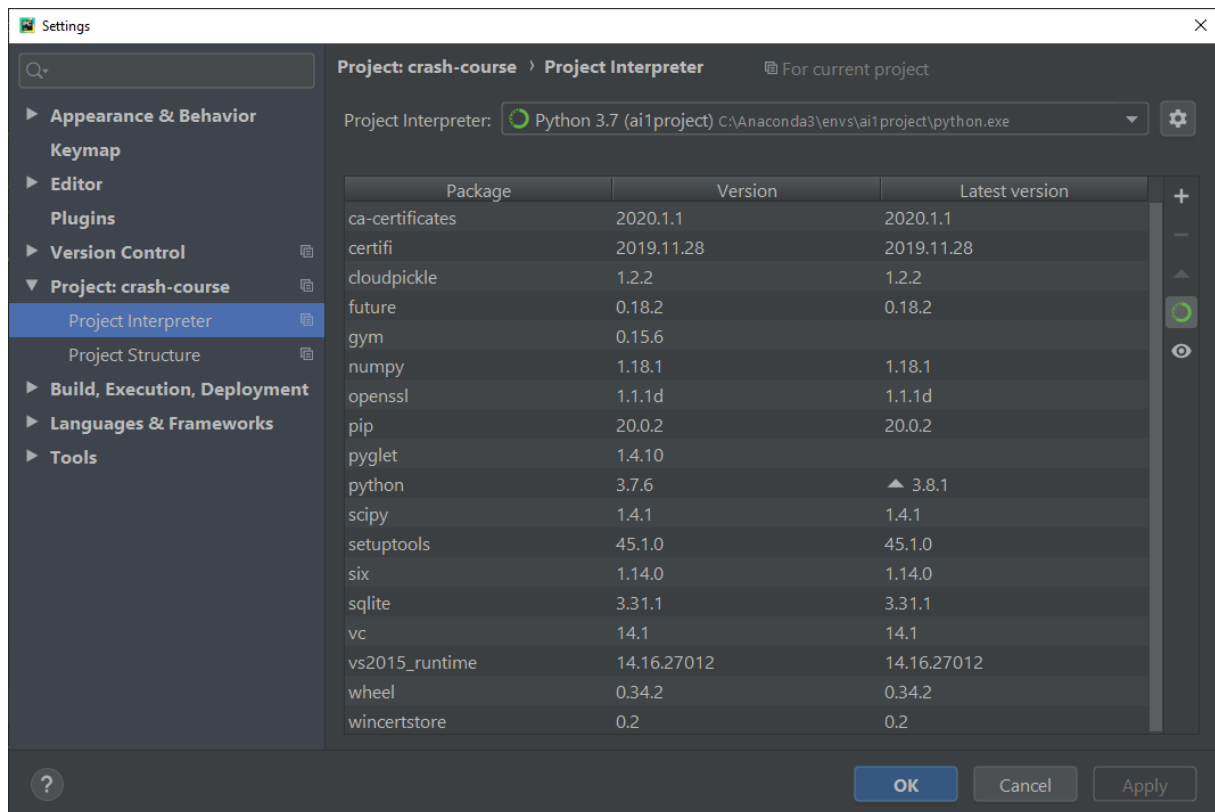


Er wordt automatisch een nieuw project “crash-course” in de opgegeven directory aangemaakt. Je kan voor “New window” kiezen om het in een nieuw venster te openen.

Stap 2: Interpreter configureren:

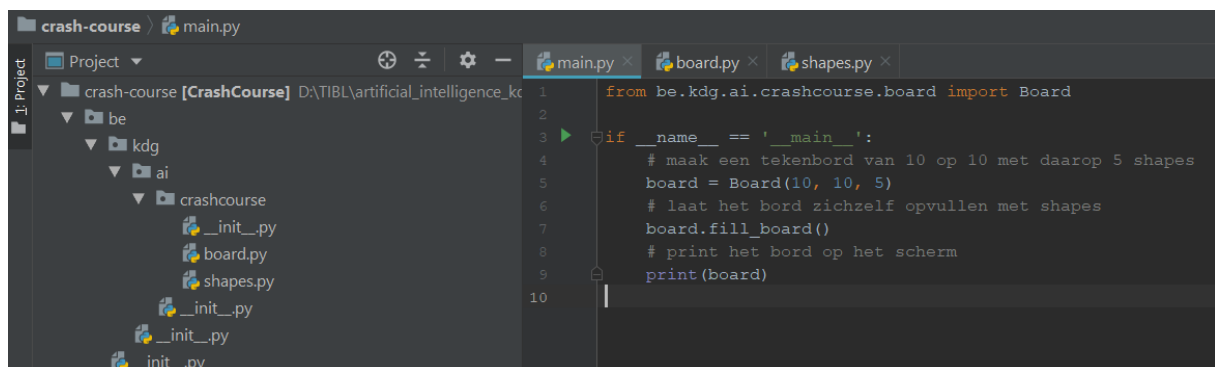
Je krijgt de melding dat er geen interpreter is geconfigureerd. We gaan de python.exe van Anaconda gebruiken.

Via “File > Settings...” openen we het “Settings” venster. We gaan naar “Project: crash-course > Project Interpreter” (links) en rechts bovenaan kiezen we voor de python exe van onze ai1project conda omgeving:



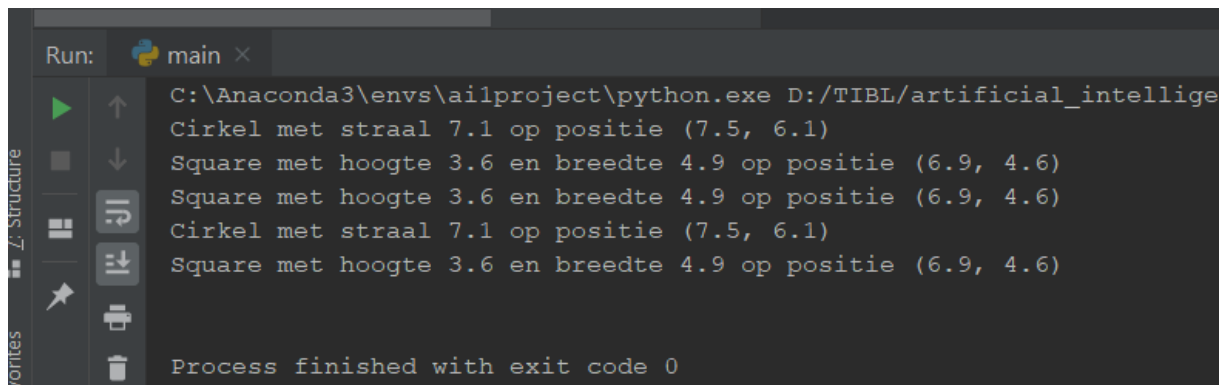
Stap 3: voorbeeldproject bekijken:

We openen nu de files main.py, board.py en shapes.py:



En we klikken op het groene “Run” pijltje op lijn 3 van main.py.

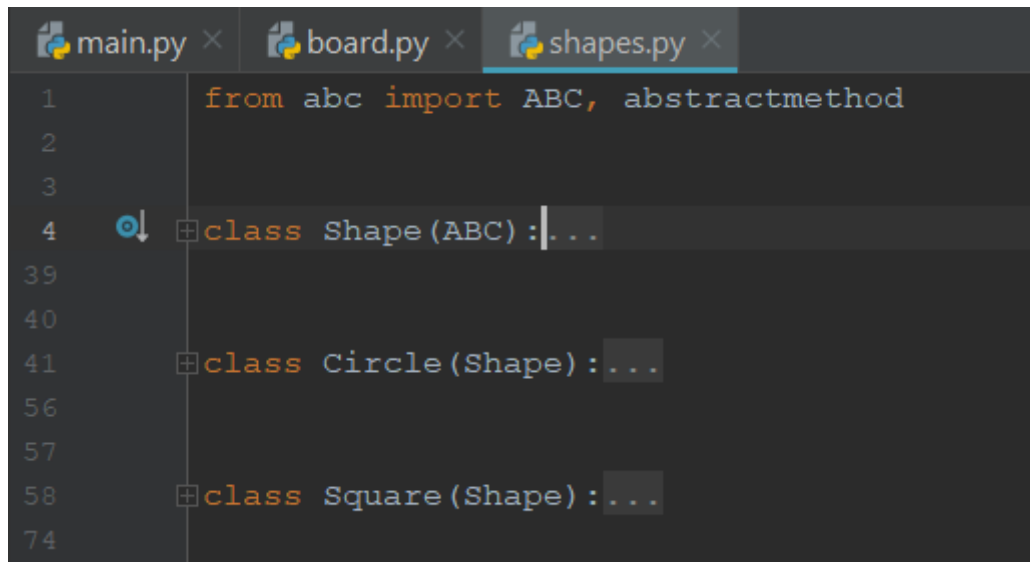
De Python console wordt automatisch geopend en we krijgen:



```
Run: main ×
C:\Anaconda3\envs\ailproject\python.exe D:/TIBL/artificial_intelligence
Cirkel met straal 7.1 op positie (7.5, 6.1)
Square met hoogte 3.6 en breedte 4.9 op positie (6.9, 4.6)
Square met hoogte 3.6 en breedte 4.9 op positie (6.9, 4.6)
Cirkel met straal 7.1 op positie (7.5, 6.1)
Square met hoogte 3.6 en breedte 4.9 op positie (6.9, 4.6)
Process finished with exit code 0
```

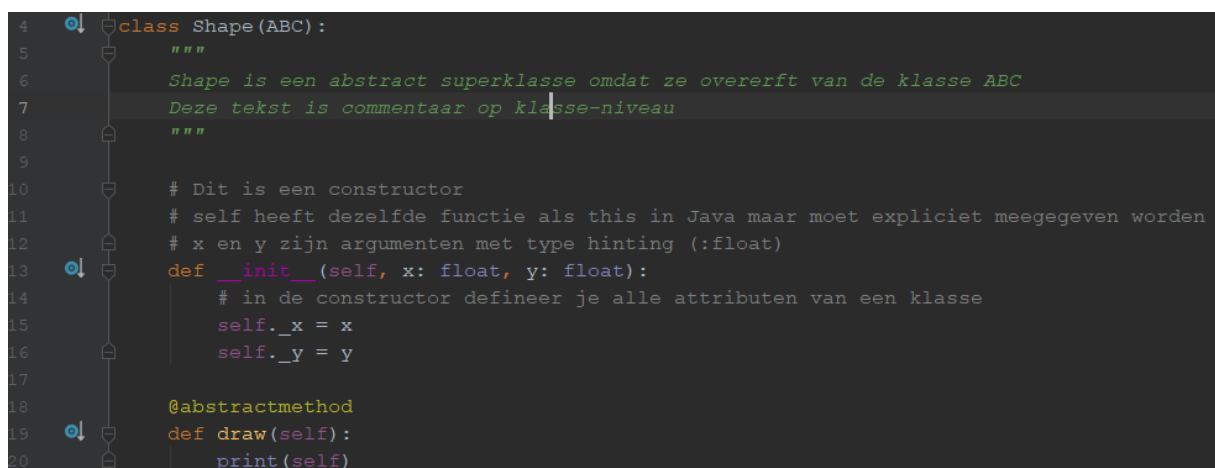
Het is dus gelukt om de code te runnen.

In shapes.py wordt de package “abc” gebruikt om een abstracte superklasse “Shape” te definiëren waarvan de klassen “Circle” en “Square” erven. “Shape” zelf erft van klasse “ABC” uit de package “abc”. De superklassen waarvan een klasse erft worden dus tussen haakjes geplaatst na de naam van de klasse in de “class” definitie.



```
main.py × board.py × shapes.py ×
1 from abc import ABC, abstractmethod
2
3
4 class Shape(ABC): ...
39
40
41 class Circle(Shape): ...
56
57
58 class Square(Shape): ...
74
```

Klasse “Shape” bevat een abstracte methode “draw” die door de subklassen moet geïmplementeerd worden:



```
4 class Shape(ABC):
5     """
6     Shape is een abstract superklasse omdat ze overerft van de klasse ABC
7     Deze tekst is commentaar op klasse-niveau
8     """
9
10    # Dit is een constructor
11    # self heeft dezelfde functie als this in Java maar moet expliciet meegegeven worden
12    # x en y zijn argumenten met type hinting (:float)
13    def __init__(self, x: float, y: float):
14        # in de constructor definieer je alle attributen van een klasse
15        self._x = x
16        self._y = y
17
18    @abstractmethod
19    def draw(self):
20        print(self)
```

In board.py zien we hoe properties worden gedefinieerd via backing fields. Property “width” in klasse “Board” heeft een backing field “self._width” (die private is, vandaar de underscore) die in de constructor “__init__” wordt gedefinieerd. De eigenlijke property wordt gecreëerd door een functie “width” die de waarde van “self._width” retourneert:

```
7 class Board:
8     """
9     Deze klasse stelt een tekenbord voor waarop enkele shapes geplaatst worden
10    """
11
12    # n is the number of shapes
13    def __init__(self, width: float, height: float, n: int):
14        self.shapes = None
15        self._n = n
16        self._width = width
17        self._height = height
18        self.shape_generator = ShapeGenerator(self)
19
20    # deze methode maakt gebruik van een list comprehension en generators
21    def fill_board(self) -> None:
22        self.shapes = [shape for shape in self.shape_generator]
23
24    @property
25    def random_location(self) -> tuple:
26        return random.uniform(0, self._width), random.uniform(0, self._height)
27
28    @property
29    def width(self) -> float:
30        return self._width
```

De klasse “ShapeGenerator” erft van “Iterable” zodat hij als generator kan gebruikt worden:

```
47 class ShapeGenerator(Iterable):
48     """
49     Deze klasse implementeert een iterator zodat hij gebruikt kan worden als generator
50
51     """
52
53    def __init__(self, board: Board):...
54
55    def __iter__(self):...
56
57    def __next__(self) -> Shape:...
58
59    def create_circle(self) -> Circle:...
60
61    def create_square(self) -> Square:...
```