**Van Java naar Python – Cheatsheet**

**Basisoperaties**

De onderstaande tabellen beschrijven hoe je verschillende programmeertechnieken uit Java omzet naar Python. In de linkerkolom wordt telkens een Java-voorbeeld gegeven, en in de rechterkolom de bijbehorende Pythonvertaling. Bij alle Java-coderegels waar geen main()-methode bij wordt vermeld, mag je ervan uitgaan dat ze wel degelijk in de main()-methode geschreven zijn.

Lees per tabelregel de bijbehorende toelichting goed door! Deze geeft namelijk waar nodig extra uitleg over de verschillen tussen Python en Java.

|  |  |
| --- | --- |
| **In Java schrijf je…** | **, maar in Python schrijf je…** |
| System.out.println("Hello World");  //Dit is een comment | print("Hello World")  #Dit is een comment |
| String naam = "Jelle"; | naam = "Jelle"  #Zie toelichting: punt 1 |
| Scanner scanner = new Scanner(System.in);  int leeftijd = scanner.nextInt(); | Leeftijd = int(input())  #Zie toelichting: punt 2 |
| int aantalDiamanten = 30;  String tekst = aantalDiamanten + " is een mooi aantal diamanten!"; | aantalDiamanten = 30  tekst = f'{aantalDiamanten} is een mooi aantal diamanten!'  #Zie toelichting: punt 3 |
| boolean b1 = true;  boolean b2 = false;  boolean and\_combi = b1 && b2;  boolean or\_combi = b1 && b2;  boolean b1\_omgedraaid = !b1; | b1 = True  b2 = False  and\_combi = b1 and b2  or\_combi = b1 or b2  b1\_omgedraaid = not b1  #Zie toelichting: punt 4 |

**Toelichting**

1. Het is Pythontechnisch niet nodig om datatypen van variabelen te definiëren.
2. In dit geval is converteren naar een int nodig, omdat we een geheel getal willen inlezen. Stel dat we een string hadden willen inlezen, dan had je int() weg moeten laten.
3. Eén van de lastiger dingen van Python is dat je aan Stringconcatenaties niet zomaar variabelen van een ander datatype, bijvoorbeeld int, kan toevoegen. Het gebruik van de standaard + om een String en een ander datatype achter elkaar te plakken resulteert in een error. Door op deze manier f en {} te gebruiken krijg je deze concatenatie wél voor elkaar (te vergelijken met het gebruik van printf in Java). Je hoeft trouwens geen ; neer te zetten bij meerdere coderegels: Python detecteert zelf wanneer elk statement begint en eindigt.
4. true en false in Java, zijn True en False in Python (let dus op het hoofdlettergebruik). Daarnaast mag je in Python alle logische operatoren (and or en not) in het Engels opschrijven.

|  |  |
| --- | --- |
| if("Jelle".equals("Jelle")) {  System.out.println("'Jelle’ is inderdaad gelijk aan 'Jelle'");  }  else if(5 > 3) {  System.out.println("5 is inderdaad groter dan 3!");  }  else {  System.out.println("Nu kan ik niks meer bedenken…");  } | if("Jelle" == "Jelle"):  print("'Jelle' is inderdaad gelijk aan 'Jelle'")  elif(5 > 3):  print("5 is inderdaad groter dan 3!")  else:  print("Nu kan ik niks meer bedenken…")  #Zie toelichting: punt 1 |
| int i = 0;  while(i < 10) {  System.out.println(i);  i++;  } | i = 0  while(i < 10):  print(i)  i += 1  #Zie toelichting: punt 2 |
| for(int i = 0; i < 10; i += 2) {  System.out.println(i);  } | i = 0  for i in range(0, 10, 2):  print(i)  #Zie toelichting: punt 3 |

**Toelichting**

1. In tegenstelling tot Java zijn in Python tabjes héél belangrijk. Als je bijvoorbeeld een bepaalde if(), while() of functie in gaat moét je een tab gebruiken om een volgend detailniveau van je code te definiëren.
2. ++ en -- werken wel in Java, maar niet in Python. In plaats daarvan moet je += 1 en -=1 gebruiken.
3. De for-loop werkt in Python heel anders dan in Java. Ten eerste definieer je de iterator altijd vóórdat je deze loop opstart. Vervolgens loop je met deze iterator een range langs, die 3 parameters ontvangt. De eerste staat voor de startwaarde, de tweede voor de eindwaarde en de derde voor de stapwaarde.

|  |  |
| --- | --- |
| public static String geefBegroeting(String naam) {  return "Hallo " + naam + "!";  }  public static void main(String[] args) { System.out.println(geefBegroeting("Jelle");  } | def geefBegroeting(naam):  return "Hallo " + naam + "!"  print(geefBegroeting("Jelle"))  #Zie toelichting: punt 1 |
| int mijnLeeftijd\_int = 28;  double mijnLeeftijd\_double = (double) mijnLeeftijd\_int;  mijnLeeftijd\_double += 0.5;  System.out.println(mijnLeeftijd\_double); | mijnLeeftijd = 28  mijnLeeftijd = mijnLeeftijd + 0.5  print(mijnLeeftijd)  #Zie toelichting: punt 2 |

**Toelichting**

1. Met het keyword def kan je in Python alle functies aanmaken.
2. Typecasting in Python gaat iets eenvoudiger dan in Java: je kan namelijk van een variabele het datatype wijzigen, zonder dat je daarvoor een nieuwe variabele aan moet maken.

|  |  |
| --- | --- |
| public class Dier {  private String naam;  private int leeftijd;  public Dier(String naam, int leeftijd) {  this.naam = naam;  this.leeftijd = leeftijd;  }  }  public class Hond extends Dier {  private String reu\_of\_teef;  public Hond(String naam, int leeftijd, String reu\_of\_teef) {  super(naam, leeftijd);  this.reu\_of\_teef = reu\_of\_teef;  }  public void maakGeluid() {  System.out.println("Woef!");  }  }  public class Kat extends Dier {  private int aantalSnorharen;  public Kat(String naam, int leeftijd, int aantalSnorharen) {  super(naam, leeftijd);  this.aantalSnorharen = aantalSnorharen;  }  public void maakGeluid() {  System.out.println("Miauw!");  }  }  public class Main {  public static void main(String[] args) {  Hond hond = new Hond("Bello", 1, "reu");  Kat kat = new Kat("Poekie", 2, 8);  hond.maakGeluid();  kat.maakGeluid();  }  } | class Dier:  def \_\_init\_\_(self, naam, leeftijd):  self.naam = naam  self.leeftijd = leeftijd  class Hond(Dier):  def \_\_init\_\_(self, naam, leeftijd, reu\_of\_teef):  super().\_\_init\_\_(naam, leeftijd)  self.reu\_of\_teef = reu\_of\_teef    def maakGeluid(self):  print("Woef!")  class Kat(Dier):  def \_\_init\_\_(self, naam, leeftijd, aantalSnorharen):  super().\_\_init\_\_(naam, leeftijd)  self.aantalSnorharen = aantalSnorharen      def maakGeluid(self):  print("Miauw!")  hond = Hond("Bello", 1, "reu")  kat = Kat("Poekie", 2, 8)  hond.maakGeluid()  kat.maakGeluid()  #Zie toelichting |

**Toelichting**

Objectoriëntatie werkt in Python heel anders dan in Java. Allereerst maak je een constructor door \_\_init\_\_ op te schrijven, en deze in elk geval de parameter ‘self’ mee te geven. Dit keyword geeft namelijk aan dat de methode wordt opgeroepen op een instantie van de betreffende klasse. Daarnaast maak je alle klasse-attributen niet búiten, maar juist ín deze constructuur aan, eveneens met behulp van ‘self’. Ook kan je inheritance aangeven door tussen haakjes achter een klasse de naam van de betreffende superklasse te schrijven. Hierbij ben je wel verplicht om aan de \_\_init\_\_()-methode alle parameters mee te geven die gebruikt worden in de superklasse en deze vervolgens te gebruiken om de \_\_init\_\_()-methode van die superklasse aan te roepen met behulp van super(). Ten slotte bestaan er in Python geen private-variabelen en ook geen interfaces: het is van oorsprong namelijk geen objectgeoriënteerde programmeertaal.

**Verzamelingen**

Python kent een aantal datastructuren waarin verschillende soorten data opgeslagen kunnen worden. De onderstaande tabel vat het gebruik hiervan samen.

|  |  |  |  |
| --- | --- | --- | --- |
| **Datastructuur** | **Beschrijving** | **Kenmerkend(e) symbo(o)l(en)** | **Codevoorbeeld** |
| List (‘list’) | Vergelijkbaar met Java-arrays. Op basis van indexnummers kunnen lijstelementen aangeroepen én aangepast worden. | Vierkantshaken bij het aanmaken én aanroepen:  [ en ] | mijn\_lijst = [1, 2, 3]  print(mijn\_lijst[0])  mijn\_lijst[0] = 2  print(mijn\_lijst[0]) |
| Tupel (‘tuple‘) | Vergelijkbaar met lists en Java-arrays, maar aanpassingen zijn nu niet mogelijk!  De rode regel in het codevoorbeeld resulteert daarom in een error. | Ronde haken bij het aanmaken:  ( en )  Vierkantshaken bij het aanroepen:  [ en ] | mijn\_tupel = (1, 2, 3)  print(mijn\_tupel[0])  mijn\_tupel[0] = 2  print(mijn\_tupel[0]) |
| Dictionary (‘dict‘) | Hierin gebruik je geen indexen om bij de ‘values’ te komen, maar zelfbedachte waarden (‘keys’). Elke key wordt dus gekoppeld aan één waarde. | Accolades bij het aanmaken:  { en }  Dubbele punten bij het onderscheiden van keys en values  Vierkantshaken bij het aanroepen:  [ en ] | mijn\_dict = {  'plaats' : 'Leiden',  'inwoners' : 121562  }  print(mijn\_dict['inwoners'])  mijn\_dict['inwoners'] += 1  print(mijn\_dict['inwoners']) |