PROGRAMMEREN 1

LES 1: INTRODUCTIE

Download PDF-versie

UCLL HOGESCHOOL

OVERZICHT

- Kennismaking
- Over dit vak
- Wat is programmeren?
- Eerste programma
- Expressies
- Datatypes
- Invoer & Uitvoer
- Introductie tot modules

KENNISMAKING

OVER MIJ

- ir. Thomas Vranken
- Computerwetenschappen aan de KU Leuven
- Educatieve master Wetenschappen en Technologie aan de KU Leuven
- Software engineer bij Twipe Mobile Solutions
- Lesgever bij CodeFever

OVER JULLIE

- Naam?
- Specialisaties binnen de lerarenopleiding?
- Andere relevante ervaringen voor dit vak? Jobstudent, werkervaring, voorgaande opleidingen, ...

OVER DIT VAK

DOELSTELLINGEN VAN DIT VAK

ECTS-fiche

DE STUDENT VERWERFT DE BASISKENNIS VOOR PROGRAMMEREN:

- Programma's kunnen schrijven in meerdere programmeertalen
- Een aantal belangrijke programmeerconcepten kunnen uitleggen en toepassen
- Programmeerconcepten kunnen combineren om nieuwe problemen op te lossen
- Computationeel denken kunnen uitleggen aan de hand van voorbeelden

DE STUDENT LEERT EEN AANTAL PRINCIPES VOOR VAKDIDACTISCH HANDELEN:

- Correcte terminologie en taalgebruik kunnen benutten
- Abstracte concepten en denkprocessen kunnen verbinden met voorwerpen uit de realiteit

LEERSTOF

- Slides zijn voornaamste leerstof
- Handboek: De Programmeursleerling: Leren coderen met Python 3 van Pieter Spronck
  - Link ook op Toledo > Cursusdocumenten
  - Niet alles wordt behandeld
  - Vooral nuttig als referentiewerk + extra oefeningen
- Oefeningen op Dodona

Zelf (online) kunnen zoeken naar informatie, is cruciaal voor een programmeur

- Je zal oefeningen moeten (her)maken in Kotlin, een taal die niet expliciet wordt behandeld tijdens de les
- Je zal kleinere opdrachten krijgen om andere talen zelf te analyseren

EVALUATIE

- Opdrachten indienen tijdens het jaar
  - Worden meegedeeld tijdens de les + via Toledo
  - Indienen via Dodona
  - Meer info op Toledo > Opdrachten
- Project maken
  - Individueel
  - Mondelinge verdediging tijdens examenperiode
  - Opdracht komt online tijdens/na de paasvakantie

Er is **geen** theoretisch examen tijdens de examenperiode.

PRAKTISCH

- 9-tal lessen van 3u op woensdagnamiddag
- On-campus (tenzij anders meegedeeld)
- Doel is om merendeel van theorie vóór de paasvakantie te zien
  - Ruimte om nadien aan project te werken en vragen te stellen

SOFTWARE EN PROGRAMMEEROMGEVINGEN

- Jullie gaan met verschillende programmeeromgevingen in contact komen
- Doel is om ermee kennis te maken, niet om diepgaan mee vertrouwd te geraken
- Deze les: https://replit.com/
  - Normaal al account gemaakt
  - Indien niet: nog kans deze les

TIPS OM TE STUDEREN

Oefening baart kunst:

- Programmeren leer je **niet** door alleen maar code te lezen, maar door ze zelf te schrijven
- Belangrijkste is om de concepten en het denkproces onder de knie te krijgen
  - Afkijken van oplossingen heeft weinig zin
- Veel oefeningen (her)maken is nuttig

Je moet geen grote hoeveelheden theorie instuderen, maar je moet wel vlot code kunnen schrijven. Een aantal concepten leer je daarom best vanbuiten.

WAT IS PROGRAMMEREN?

VOORBEELDEN VAN PROGRAMMEERTALEN(?)

Welke van onderstaande talen zijn geen programmeertalen?

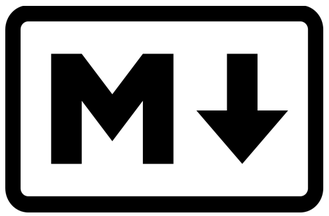
Niet-programmeertalen lichten rood op wanneer je jouw muis er over houdt

Scratch

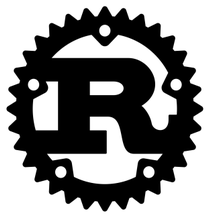
JavaScript

Python

Java



Markdown



Rust

C++

HTML

Kotlin

PROGRAMMEERTALEN

Programmeren = instructies geven aan een computer om een probleem op te lossen

- *Probleem* is hier een zeer ruim begrip

Programmeertaal = een set van regels waarmee we met de computer kunnen communiceren om deze problemen op te lossen

- Is meestal een middel, geen doel
- In theorie kan een probleem opgelost worden met eender welke programmeertaal
- Een aantal concepten bestaan in bijna alle programmeertalen → onderwerpen van dit vak

HTML en Markdown zijn nuttige en veelgebruikte talen, maar ze ondersteunen niet al deze concepten.

EERSTE PROGRAMMA: HELLO WORLD

```python
print("Hello, world!")
```

Python

HELLO WORLD

```python
print("Hello, world!")
```

- `print()` is een **functie** in Python die iets toont in de uitvoer (de console)
- Hetgene tussen de haakjes noemt men een **parameter**
- `"Hello world"` noemt men een `string`: een stuk tekst (te herkennen aan de aanhalingstekens)
- Computers geven geen interpretatie aan een `string`. Het is betekenisloos, in tegenstelling tot `print()`

'Hello world' is traditioneel het eerste programma dat een programmeur probeert te schrijven wanneer die een nieuwe programmeertaal leert. Zie bv. Hello, World op Wikipedia.

OPDRACHT: HELLO WORLD

- Log in op https://replit.com/ en maak een project voor Python
- Schrijf een lijn code in de Python **Console** (rechterkant van het project) om je eigen naam als `string` te printen en voer uit
- Verwijder de aanhalingstekens rond je naam en voer de code opnieuw uit. Wat gebeurt er? Waarom?

CONSOLE EN PROGRAMMA'S

- De console is een interactieve omgeving, bedoelt voor korte testen en code van 1 lijn uit te voeren
  - Je 'praat' rechtstreeks met de computer
- Alles in de console ben je kwijt als de console wordt gesloten (bv. door het project te verlaten)
- Een programma is een reeks instructies die de computer uitvoert
- Een programma kan meerdere lijnen code bevatten. Elke lijn stelt een nieuwe instructie voor(*)
- Een programma kan je opslaan en later opnieuw uitvoeren (eventueel op een andere computer)

(*) dit is wat kort door de bocht, de details komen in volgende lessen aan bod

OPDRACHT: RECEPT

Schrijf een **programma** dat een recept naar keuze uitprint:

- Er moeten minstens 3 en maximaal 6 stappen zijn
- Schrijf voor elke stap een aparte `print()` op een nieuwe lijn
- Nummer elke stap door een cijfer in het begin van de instructie te zetten

Als je programma werkt en de correcte uitvoer toont, wissel dan twee lijnen code van plaats. Voer de code opnieuw uit en bekijk de uitvoer. Wat besluit je hieruit?

VOLGORDE VAN UITVOEREN

De basisregel is:

- De computer start op lijn 1 van het programma
- De computer voert lijn per lijn uit van boven naar beneden

Dit is slechts een basisregel. Er bestaan verschillende uitzonderingen op!

EXPRESSIES

```Python
> (2 + 7) * 4
36

> "Welkom aan de" + "UCLL!"
Welkom aan deUCLL!
```

EXPRESSIES: BASIS

- Een **expressie** kan je **evalueren**. De computer leest de aparte waarden en combineert ze volgens welbepaalde regels. Een expressie heeft altijd een **resultaat** (een nieuwe waarde)
- Een expressie kan wiskundig zijn (bv. `42`, `2 + 7`), maar andere soorten expressies bestaan ook (bv. `"Welkom aan de"`)
- Het resultaat van een expressie kan je snel te weten komen door de expressie te evalueren in de **console**

Bekijk de tweede expressie op de vorige slide opnieuw. Wat valt je op? Hoe kan je dit oplossen?

EXPRESSIES: WISKUNDE

De volgende bewerkingen zijn universeel over (bijna) alle programmeertalen:

- Optellen `+`
- Aftrekken `-`
- Vermenigvuldigen `*`
- Delen `/`
- Modulo (rest na deling) `%`

Programmeertalen respecteren de volgorde van bewerkingen:

- Bijvoorbeeld: `*` en `/` hebben voorrang op `+` en `-`
- Gebruik extra haakjes `(` en `)` indien nodig

Andere bewerkingen (machten, wortels, logaritmes, ...) variëren tussen programmeertalen.

EXPRESSIES: TEKST

Je kan twee of meerdere `strings` samenvoegen door middel van `+`. Dit is redelijk universeel voor programmeertalen, op een paar randgevallen na.

Men noemt dit ook wel **concateneren** (*Eng: concatenate*), vaak afgekort tot *concat*.

Wees expliciet:

- Computers plaatsen normaal **geen** spaties of witregels, de programmeur moet dit toevoegen
- Spaties zijn eenvoudig: plaats het binnen een `string`, of concat `" "` met een andere `string`
  - Bijvoorbeeld: `"Hello " + "World"` of `"Hello" + " " + "World"`
- Witregels komen later aan bod

VARIABELEN

```python
vak = "Programmeren 1"
print(vak)
```

Python

VARIABELEN: BASIS

Een **variabele** dient voor het **opslaan** van expressies en andere data.

- Een variabele heeft steeds een **naam** en een **waarde**
- **declareren**: het aanmaken van een nieuwe variabelen
- **toekennen**: een (bestaande of nieuwe) variabele een waarde geven

In Python gebeurt declaratie en toekenning tesamen in één lijn. De algemene syntax is:

```
<naam_variabele> = <waarde_variabele>
```
Python

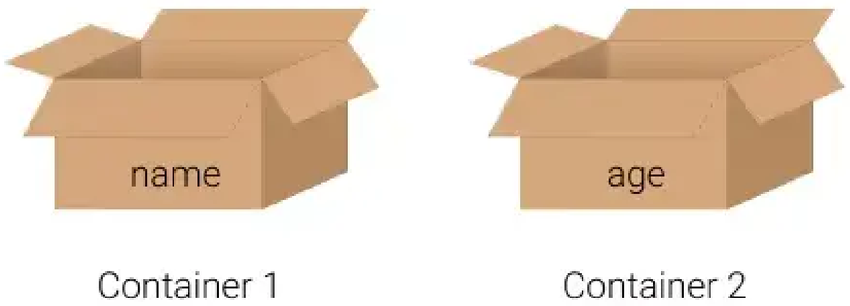
Dit geldt niet algemeen. In sommige talen kan je een variabele eerst declareren, en pas veel later een waarde geven. Bijvoorbeeld in JavaScript:

```
let naam;  // declaratie
// wat code tussendoor...
naam = "Bob"  // toekenning
```
JavaScript

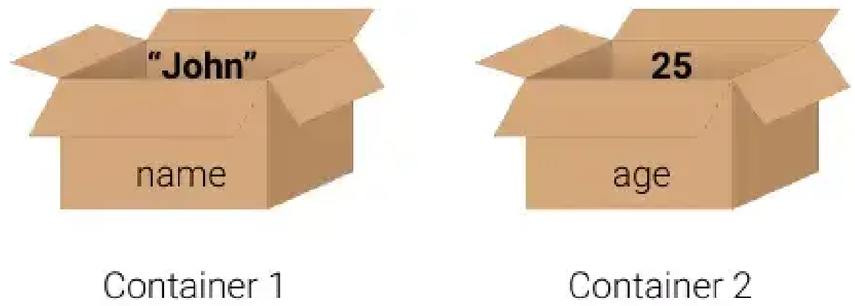
MENTAAL MODEL

Vaak wordt een variabele vergeleken met een container:

- De naam is een label waarmee we de container kunnen terugvinden
- De waarde is de inhoud van de container
- Declareren → een nieuwe container maken
- Een waarde toekennen → de container vullen



Variabelen declararen



Variabelen een waarde toekennen

Let op: een variabele in programmeren is iets helemaal anders dan een variabele in wiskunde!

VARIABELEN: GOEDE GEWOONTES

Vergelijk de twee stukken code hieronder. Welke is het eenvoudigst om te begrijpen?

```python
x = 20
y = "An"
z = "V"
```

```python
leeftijd = 20
naam = "An"
geslacht = "V"
```

Geef variabelen steeds een duidelijke naam. Dit verhoogt de leesbaarheid van code.

CONVENTIES

Verschillende programmeertalen hebben andere gewoontes voor de namen van variabelen.

In Python gebruikt men **snake_case**:

- Gebruik enkel kleine letters
- Vervang spaties door een laagliggend streepje (_)

Voorbeelden: `gemiddelde_score`, `een_zeer_lange_variabelenaam`, `ucll_aantal_lln`

Een populair alternatief in talen zoals Java is **camelCase**:

- Start met een kleine letter, tenzij het een afkorting is in hoofdletters
- Vervang spaties door de volgende letter als een hoofdletter te schrijven

Voorbeelden: `gemiddeldeScore`, `eenZeerLangeVariabelenaam`, `UCLLAantalLln`

CONVENTIES

Maakt het uit welke je notatie je gebruikt?

- Meestal niet
- Sommige talen dwingen het wel af of suggereren om de naam aan te passen
- Het belangrijkste is dat je consistent bent

VARIABELEN EN EXPRESSIES

Een variabele kan het resultaat van een expressie bevatten, of zelfs de waarde van een andere variabele.

Bijvoorbeeld:

```python
straal = 5
oppervlakte = straal * straal * 3.14
print(oppervlakte)

gebruiker = "Bart"
begroeting = "Welkom " + gebruiker
print(begroeting)
```

Wat gebeurt er precies?

1. De computer evalueert eerste de expressie aan de rechterkant van `=`
2. Het resultaat wordt gekopieerd naar de variabele aan de linkerkant van `=`

DE WAARDE VAN EEN VARIABELE VERANDEREN

Je kan bestaande variabelen een nieuwe waarde geven. In Python gebeurt dit op dezelfde manier als het declareren van een nieuwe variabele:

```python
dag = "maandag" # Dit is een declaratie van een nieuwe variabele
# ...
dag = "dinsdag" # Hier krijgt een bestaande variabele een nieuwe waard
```

Opgelet: In Python kan je niet aan de code zien of een variabele nieuw is of al bestaat. Dit kan een nadeel zijn bij het lezen van complexere programma's.

Andere talen hebben regels om het verschil duidelijk te maken. Bijvoorbeeld in JavaScript:

```javascript
let dag = "maandag" # dit is een declaratie
# ...
dag = "dinsdag" # dit is geen declaratie, de variabele moet al bestaan
```

QUIZ

Bekijk de code hieronder. Wat zal er in de console verschijnen?

```python
t = 5
y = t + 3
t = 6
print(t)
print(y)
```

5 en 8

11 en 9

6 en 6

6 en 8

AANPASBAARHEID

Variabelen kunnen aanpasbaar *(Eng: mutable)* of onaanpasbaar *(Eng: immutable)* zijn.

In Python zijn alle variabelen steeds mutable. Je kan — helaas — niet afdwingen dat een variabele immutable is. Andere talen hebben strictere regels.

Over het algemeen werk je best zoveel mogelijk met immutable variabelen. Hoe vaker je variabelen muteert (aanpast), hoe moeilijker het meestal wordt om code te begrijpen.

CONSTANTEN

Sommige variabelen stellen een constante waarde voor in het hele programma. Voorbeelden: het getal π, het aantal levens in het begin van een spel, het aantal dagen in een week, ...

Constanten zijn logischerwijs onaanpasbaar. Om dit aan te geven, wordt hun naam vaak geschreven in `UPPER_SNAKE_CASE`.

Bijvoorbeeld:

```python
DAYS_IN_WEEK = 7
```

Dit is een afspraak tussen mensen, het is geen verplichting. Computers houden hier geen rekening mee.

VARIABELEN: OPDRACHT

Schrijf het volgende programma:

- Declareer een variabele die je geboortejaar voorstelt
- Declareer een constante die de leeftijd voorstelt waarop iemand als meerderjarig wordt beschouwd (18 jaar)
- Declareer een nieuwe variabele die het jaar voorstelt waarop je meerderjarig wordt/bent geworden
- Print het jaar van meerderjarigheid in de console
- Pas nadien het geboortejaar aan naar keuze. Controleer of je code blijft werken zonder verdere aanpassingen

DATATYPES

```python
naam = "Chiara"
leeftijd = 53
boodschap = "Naam: " + naam + ", leeftijd: " + leeftijd

=> TypeError: can only concatenate str (not "int") to str
```

DATATYPES: BASIS

Elke variabele en expressie heeft een bepaald **type**, afhankelijk van het soort data dat het voorstelt.

De meest voorkomende types zijn:

- Tekst: `string`
- Gehele getallen: `int`, afgeleid van *Integer*
- Kommagetallen: `float`, afgeleid van *Floating point numbers*
- Booleaanse waarden: `bool`, zie volgende les

De namen hierboven kunnen variëren tussen programmeertalen.

DATATYPES: NUT

Types hebben verschillende toepassingen:

- Afleiden welke operaties wel/niet mogelijk zijn
  - Bijvoorbeeld: een `+` heeft een andere functie bij tekst dan bij cijfers
- Programmeurs waarschuwen voor fouten (bugs)
  - Bijvoorbeeld: de `TypeError` in het voorbeeld waarschuwt voor het mengen van cijfers en tekst

Een simpele vuistregel: je mag niet zomaar een type mengen met een ander type ('geen appelen met peren vergelijken').

DATATYPES OPVRAGEN

In Python kan je het type van een variabele of het resultaat van een expressie opvragen met de ingebouwde functie `type()`:

```python
> type(5)
<class 'int'>

> type(dorp_of_stad)
<class 'str'>

> type(7 / 2)
<class 'float'>
```

`float` EN AFRONDINGSFOUTEN

Kommagetallen (`floats`) worden voorgesteld met een punt (geen komma).

Wat denk je dat de laatste lijn in onderstaand programma zal afprinten?

```python
print(0.1)
print(0.1 + 0.1)
print(0.1 + 0.1 + 0.1)
```

Python

- 0.1
- 0.3
- 0.6
- Iets anders

`float` EN AFRONDINGSFOUTEN

Kommagetallen (`floats`) worden voorgesteld met een punt (geen komma).

Wat denk je dat de laatste lijn in onderstaand programma zal afprinten?

```python
print(0.1)
print(0.1 + 0.1)
print(0.1 + 0.1 + 0.1)
```

- 0.1
- 0.3
- 0.6
- Iets anders

Het exacte antwoord dat Python uitprint is `0.30000000000000004`.

`float` EN AFRONDINGSFOUTEN

`float`s zijn een uitdaging voor computers:

- Je kan geen oneindige kommagetallen voorstellen met een eindig geheugen
- Ook eindige kommagetallen vormen een probleem. Hoe stel je een getal als `10 000.000 000 0001` voor met zo min mogelijk geheugen?
- Afrondingsfouten zijn haast onvermijdelijk

Conclusie: je vermijdt best kommagetallen, tenzij afrondingsfouten tolereerbaar zijn.

`float` EN AFRONDINGSFOUTEN

De naam `float` verwijst naar de interne voorstelling van kommagetallen in het geheugen van computers. De komma 'drijft' in het rond, er is geen vast aantal cijfers voor en na de komma.

Naast `float` hebben sommige talen het type `double`. Dit is een kommagetal dat meer cijfers kan bevatten dan een `float`. Het is nuttig voor toepassingen waar precisie belangrijk is (bv. voor het simuleren van het weer).

DATATYPES: HARD TYPING VS SOFT TYPING

Talen zoals Python en JavaScript gebruiken **soft typing**:

- Een variabele kan van type veranderen
- De programmeur hoeft de types niet te vermelden, ze zijn verborgen in het programma
- Types worden gecontroleerd tijdens het uitvoeren (*Eng: at runtime*)

Talen zoals Java en Rust gebruiken **hard typing**:

- Een variabele kan maar één type bevatten
- Programmeurs moeten het type (meestal) expliciet vermelden
- Types worden gecontroleerd vóór het uitvoeren (*Eng: at compile time*)

Voorbeeld uit Java:

```java
int x = 5;   // OK
String naam = 7.345   // Error
```

Hard typing is veiliger en stabieler, maar vereist meer werk en kennis van types.

DATATYPES: CONVERSIE

Elke taal heeft technieken om waarden te converteren naar een ander type.

In Python zijn er drie nuttige functies om te onthouden:

- `str()`
- `int()`
- `float()`

Bijvoorbeeld:

```python
t = str(5)
print(type(t))  # <class 'str'>

y = float("5.2")
z = int("Leuven")  # Wat gaat er gebeuren?
```

Type conversie is niet zonder risico's: gebruik ze verstandig!

INVOER EN UITVOER

INVOER EN UITVOER

Programma's op zichzelf zijn niet interessant als ze niet kunnen interageren met de 'buitenwereld'.

Kan je voorbeelden geven van interactie?

INVOER EN UITVOER

Programma's op zichzelf zijn niet interessant als ze niet kunnen interageren met de 'buitenwereld'.

Kan je voorbeelden geven van interactie?

- Muisklik of toets indrukken
- Tekst typen
- Iets tonen op het scherm
- Tekstbestanden op de computer lezen
- Het resultaat van een berekening doorsturen naar een ander proces

Dit noemt men invoer en uitvoer (*Eng: input and output*, vaak afgekort tot *I/O*)

INVOER: `input()`

```python
antwoord_str = input("Wat is je leeftijd? ")
antwoord_int = int(antwoord)
nieuwe_leeftijd_int = antwoord_int + 1
nieuwe_leeftijd_str = str(nieuwe_leeftijd_int)
boodschap = "Volgend jaar ben je " + nieuwe_leeftijd_str + " jaar oud."
print(boodschap)
```

- `input(question)` toont `question` aan de gebruiker. Het antwoord van de gebruiker wordt teruggegeven
- Je moet zelf het antwoord opslaan in een variabele
- Het antwoord is altijd van type `string`. Pas conversie van types toe indien nodig

INVOER: `input()`

SNELLE CONVERSIE

Je kan type-conversies direct toepassen op het resultaat van `input()`:

```python
antwoord_int = int(input("Wat is je leeftijd? "))
```

VEEL VOORKOMENDE FOUT

Volgende code bevat een typische fout gelinkt aan `input()`. Welke?

```python
temperatuur = input('Wat is de temperatuur in °C?')
temperatuur_kelvin = temperatuur + 273.15
```

INVOER: `input()`

SNELLE CONVERSIE

Je kan type-conversies direct toepassen op het resultaat van `input()`:

```python
antwoord_int = int(input("Wat is je leeftijd? "))
```

VEEL VOORKOMENDE FOUT

Volgende code bevat een typische fout gelinkt aan `input()`. Welke?

```python
temperatuur = input('Wat is de temperatuur in °C?')
temperatuur_kelvin = temperatuur + 273.15
```

Antwoord: Type-conversie vergeten: `temperatuur` is een `string`, geen `float`.

UITVOER: `print()`

`print()` laat toe om meer dan 1 parameter te printen:

```python
t_str = "abc"
u_int = 123
v_float = 5.236
print(t_str, u_int, v_float)
```

- elke parameter wordt apart verwerkt
- `print()` is slim genoeg om elke parameter eerst naar een `string` te converteren indien nodig
- tussen elke parameter wordt een spatie gezet in de uitvoer

INTRODUCTIE TOT MODULES

Python

```python
import math
resultaat = math.sqrt(225)
print(resultaat)
```

MODULES: STANDARD LIBRARY

Standard library = de set van functies en andere structuren die een programmeertaal altijd aanbiedt. Elke taal heeft een standard library, vaak met functies zoals `print()`, `input()`, ...

Veel talen hebben een systeem van **modules** of **packages** (betekenen vaak hetzelfde). Het laat toe om code op te splitsen over meerdere bestanden en ernaar te verwijzen.

MODULES: STANDARD LIBRARY

De Python standard library bevat een heel aantal modules. De meeste moet je eerst expliciet importeren met `import`. De officiële documentatie bevat een uitgebreide lijst met meer info over de modules en hun functies.

De volgende modules worden frequent gebruikt:

- `math`: functies voor meer complexere wiskundige bewerkingen
- `random`: genereer willekeurige getallen
- `time`: functies gerelateerd aan tijd en de voorstelling ervan

Vuistregel: plaats `import` steeds in het begin van bestanden, voordat de eigenlijke code begint.

MODULES: THIRD-PARTY LIBRARIES/MODULES/PACKAGES

Programmeurs kunnen onderling code met elkaar delen door middel van libraries/modules/packages. Rond een taal bestaat vaak een heel ecosysteem van publieke, gratis modules.

Moderne talen hebben gesofisticeerde systemen om dit te beheren. In oudere programmeertalen vereist dit meer moeite.

Afspraak voor dit vak: Third-party packages worden **niet** gebruikt voor opdrachten, noch voor het project, tenzij anders vermeld.

OPDRACHTEN

AFSPRAAK ROND VARIABELEN IN OEFENINGEN

De ervaring leert dat beginnende programmeurs moeite hebben met het beheren van variabelen en types in hun programma's.

In een strongly typed taal zoals Java wordt je verplicht hierover na te denken, omdat je types altijd expliciet moet opschrijven. Zoals eerder gezegd is Python helaas weakly typed en zal het dus niets afdwingen.

Daarom geldt in dit vak de volgende regel voor de Python-opdrachten in Dodona:

Voeg aan elke variabelenaam een postfix toe die het type aangeeft.
Bijvoorbeeld: `leeftijd_int`, `naam_str`, `temperatuur_float`, ...

OPDRACHTEN

- Ga naar Toledo > Opdrachten
- Volg de stappen om toegang te krijgen tot dit vak op Dodona
- Maak de opdrachten van module 1 in Python
- Gebruik de concepten uit deze slides en leer ze te combineren
- Je mag gebruik maken van het internet → probeer zelfstandig oplossingen te zoeken voor je problemen
- Vraag hulp als je vastzit