PROGRAMMEREN 2

TOPIC 12: GESCHIEDENIS VAN INFORMATICA EN PROGRAMMEERTALEN

Download PDF-versie

UCLL HOGESCHOOL

OVERZICHT

- Inleiding
- Hoe het begon
- Eerste programmeertalen
- Imperatieve versus declaratieve talen
- Varia

INLEIDING

Het doel van deze les is om een antwoord te bieden op een aantal vragen zoals:

- Waarom en hoe is informatica als discipline ontstaan?
- Wanneer zijn de eerste computers gebouwd en waarvoor dienden ze?
- Welke programmeertalen zijn er doorheen de jaren gebouwd?

We bekijken de geschiedenis vooral vanuit het standpunt van dit vak. Zaken zoals hardware, netwerken, ... komen minder aan bod.

1800 - 1950

HOE HET BEGON

DE INDUSTRIËLE REVOLUTIE

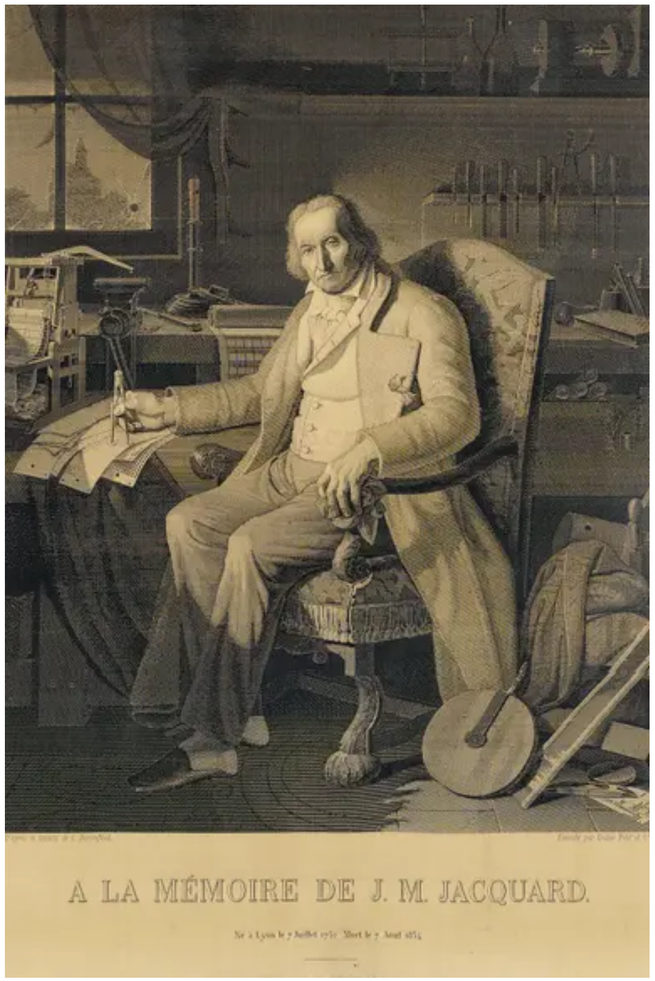
Van de 18e tot 20e eeuw veranderde de samenleving in sneltempo. Grootschalige machines, automatisatie en massaproductie werden de norm. Nieuwe uitvindingen volgden elkaar in sneltempo op.

Kenmerken van de meeste machines van die periode:

- Zijn ontworpen voor **één taak**
- Leveren **fysische** arbied en automatiseren **fysieke** taken
- Produceren een **tastbaar** resultaat

WEEFGETOUW VAN JACQUARD

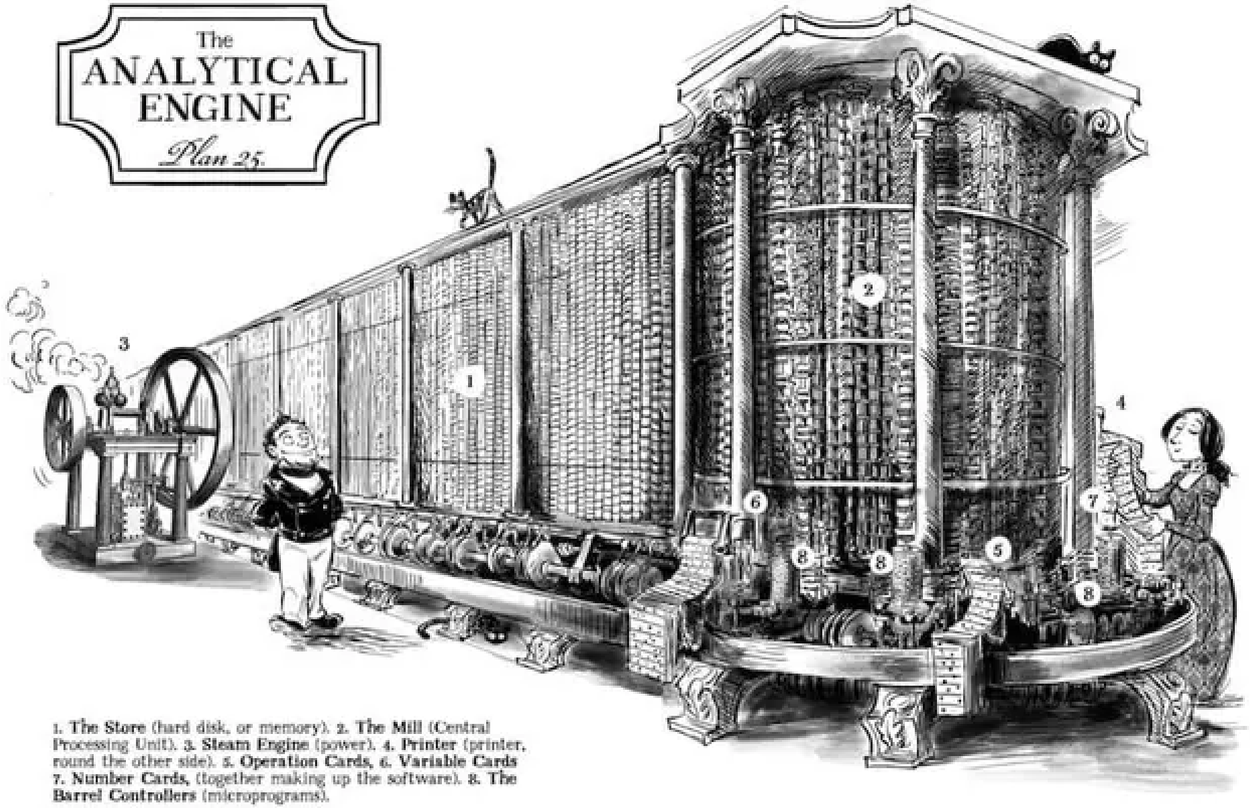
In 1804 bouwde Joseph Marie Jacquard een weefgetouw dat makkelijk meerdere soorten patronen kon weven. De machine werd aangestuurd door ponskaarten (Engels: *puching cards*). De machine was dus programmeerbaar en kon meerdere taken uitvoeren, wat uniek was voor die tijd. Links zie je een portret van Jacquard, geprogrammeerd met 24.000 ponskaarten.



CHARLES BABBAGE (1830)

Babbage wou complexe wiskundige berekeningen automatiseren. Geïnspireerd door Jacquard, bedacht hij twee machines:

- De Difference Engine: een machine om verschillende veeltermfuncties te berekenen
- De Analytical Engine: een general-purpose machine met ondersteuning voor berekeningen, iteraties en selecties, en een geheugen



1. **The Store** (hard disk, or memory). 2. **The Mill** (Central Processing Unit). 3. **Steam Engine** (power). 4. **Printer** (printer, round the other side). 5. **Operation Cards**, 6. **Variable Cards** 7. **Number Cards**, (together making up the software). 8. **The Barrel Controllers** (microprograms).

ANALYTICAL ENGINE

De Analytical Engine was Turing-compleet - nog voor de term 'Turing compleet' zelfs bestond! Ze was ontworpen om te werken met mechanische onderdelen en met stoom als energiebron. De programma's werden geschreven op ponskaarten. Ze is jammer genoeg nooit gebouwd.

De Analytical engine is een machine die:

- **meerdere taken** kan uitvoeren (eender welk wiskundig algoritme, en zelfs nog meer)
- **mentale** arbeid levert en mentale taken automatiseert
- een **'virtueel'** resultaat produceert: het genereert cijfers en letters, geen echte fysieke producten

ALAN TURING (1940)

Turing was een Engelse wiskundige en logicus. Hij bedacht ondermeer de Turing machine als manier om uit te drukken welke problemen oplosbaar zijn.

Turing leverde een aantal belangrijke bijdrage tijdens de Tweede Wereldoorlog. Hij hielp onder andere bij het kraken van Enigma, een encryptiesysteem van de Duitsers. Hiervoor werd een machine ontwikkeld die de code vele malen sneller kon kraken dan mensen. Turing ontwierp mee de machine en het algoritme. De machine had de naam 'bombe'. Ze was uitsluitend bedoeld voor decryptie, het was geen general purpose computer.

Na de oorlog werkte Turing mee aan onder andere de grondslagen van het domein Artificiële Intelligentie. Hij hield zich ook bezig met cybernetica.

KONRAD ZUSE (1940)

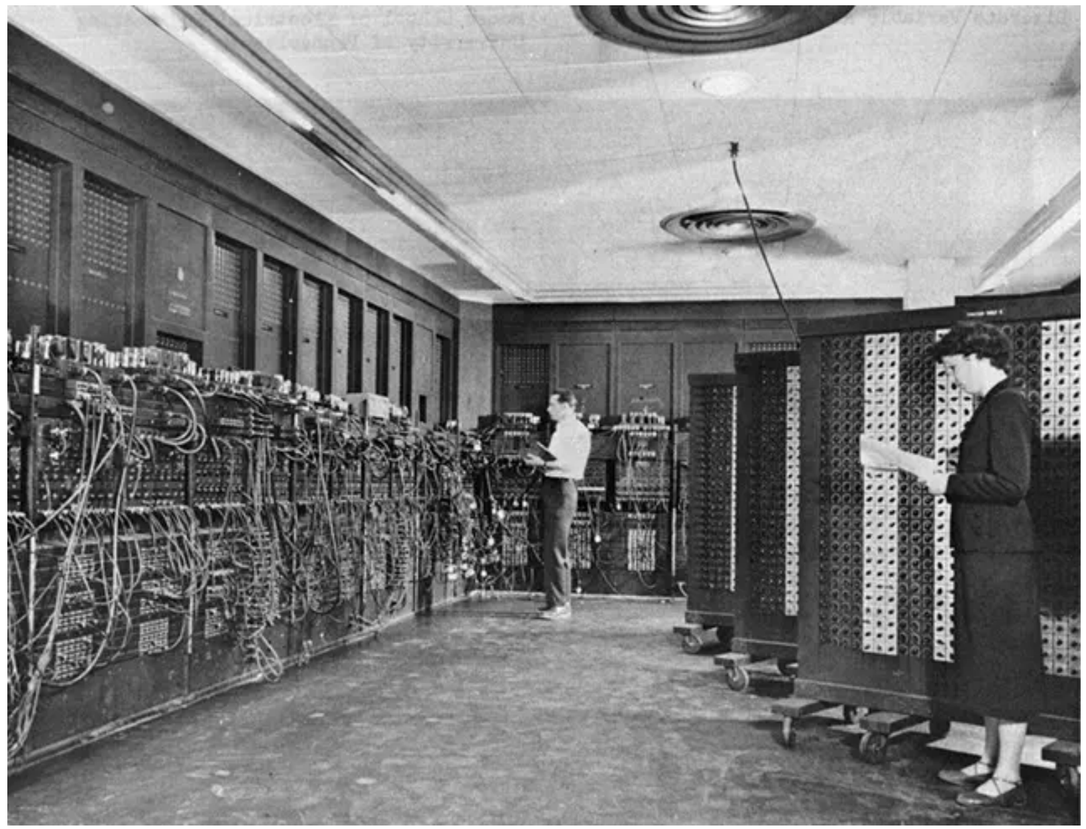
De Duitse ingenieur Konrad Zuse bouwde de eerste echt programmeerbare computer die Turing compleet was. Hij ontwierp ook de programmeertaal Plankalkül. Zuse's eerste computer was niet gebaseerd op elektronica, maar was mechanisch zoals de Analytical Engine van Babbage. Later begon hij elektronische componenten te gebruiken.

Zuse werkte grotendeels afhankelijk van andere ontwikkelaars van die tijd. Hij kende noch Turing, noch het werk van Babbage. Zijn werk bleef lang onder de radar omwille van de oorlog. Na de oorlog probeerde hij zijn computers te verkopen als bedrijf, tot het uiteindelijk werd overgekocht door IBM.

DE ENIAC (1945)

De *Electronic Numerical Integrator And Computer* werd ontworpen om berekeningen te maken. Meer specifiek zou ze hepen bij het berekenen waar artilleriegeschut zou inslaan. Ze werd gebouwd in de VS door een team onder leiding van Eckert en Mauchly, en was klaar kort na de oorlog.

De machine werd geprogrammeerd door op grote schaal kabels aan te passen op verschillende elektronische borden. Wisselen tussen twee programma's kon weken duren. Het apparaat nam een volledig lokaal in beslag.



WIE WAS EERST?

Een vaak gestelde vraag is: *Wie is de uitvinder van de computer?* Voorgaande slides tonen aan dat het antwoord genuanceerd is.

Jacquard gebruikte al verschillende fundamentele ideeën die nadien werden gekopieerd, maar bouwde geen echte general-purpose computer.

Babbage had een ontwerp met de Analytical Engine, maar heeft het nooit kunnen bouwen.

Turing lag mee aan de fundamenten van de computerwetenschappen en werkte met elektronische machines tijdens de oorlog, maar hield zich nooit echt bezig met general-purpose machines. Zijn werk werd ook pas vele jaren na de oorlog publiek gemaakt.

Zuse bouwde een echt werkende machine, maar zijn werk werd nooit opgemerkt en had daardoor beperkte inpakt op hoe computers nadien ontworpen werden.

De ENIAC wordt het vaakst genoemd als *de eerste computer*, en heeft de meeste impact gehad op de ontwikkeling van computers na de oorlog.

Computer History Museum

CHM

Menu

Explore
Overview
Stories
Timelines
Collections
Playlists
Blogs
Publications
Connect
Overview
Events
K-12 Students & Educators
Families & Community Groups
Colleges & Universities
Business & Government Leaders
Volunteers
Visit
Overview
Make a Plan
Exhibits at the Museum
Tours & Group Reservations
Customize It
Venue
About
Overview
This is CHM
Centers
Programs
News
Leadership
Join & Give
Overview
Ways to Give
Donor Recognition
Institutional Partnerships
Upcoming Events
Hours & Directions
Stories
Subscribe Now
Donate

Search for: Go

Or search the collection catalog →

Exhibit

Privacy

Today January February March April May June July August September October November December

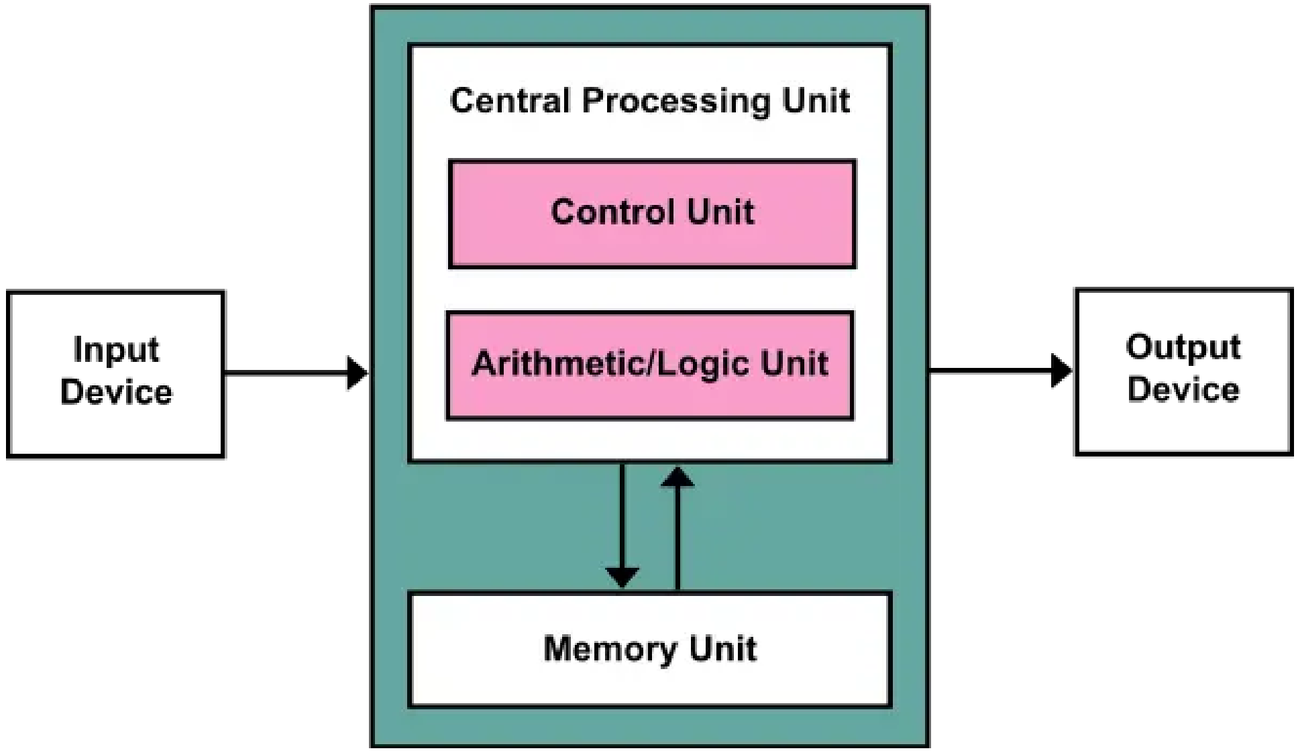
UCLL HOGESCHOOL

EERSTE PROGRAMMEERTALEN

VON NEUMANN ARCHITECTUUR

De ENIAC werd geprogrammeerd met draden en stekkerborden. Een paar jaar nadien werden computers ontwikkeld die het anders aanpakten: programma's werden symbolisch opgeslagen in het geheugen.

Een belangrijk persoon in dit verhaal was **John von Neumann**, naar wie de *von Neumann architectuur* is vernoemd. Deze architectuur wordt vandaag de dag nog steeds gebruikt.



MACHINETAAL EN ASSEMBLY

Nu programma's in geheugen voorgesteld konden worden, werden de eerste programmeertalen ontworpen om de programma's te schrijven. In machinetaal is elke instructie een binair, octaal of hexadecimaal getal. De taal hangt volledig af van de hardware en is dus anders voor elk type computer. Bovendien is ze moeilijk te lezen en te schrijven.

Assembly is een taal die een lichte abstractie biedt bovenop machinetaal. Instructies worden voorgesteld door letters en codes. Ze wordt rechtstreeks vertaald naar machinetaal. Assembly is nog steeds hardwareafhankelijk.

DE EERSTE HOGERE TALEN

FORTRAN (1958)

- Formula Translator
- ontworpen door Backus bij IBM
- bedoeld als taal voor numerieke analyse
- nog altijd (in gemoderniseerde vorm) veel gebruikt voor wetenschappelijke berekeningen

COBOL

- COmmon Business Oriented Language
- ontworpen door vrouwelijke Amerikaanse militair
- eerste records, minder wiskundig uitzicht

DE EERSTE HOGERE TALEN

LISP (1960)

- LISt Processing Language
- principe radicaal verschillend van Fortran, Cobol, ...
- taal gericht op het manipuleren van symbolische informatie in plaats van getallen of administratieve gegevens ← kunstmatige intelligentie
- John McCarthy (MIT): beginnend AI-onderzoek
- stevige wiskundige fundamenten
- Had verschillende features die later door andere talen zijn overgenomen (bv. garbage collection)

BASIC (1965)

- Beginner's All purpose Symbolic Instruction Code
- Startte als een versimpelde versie van FORTRAN voor doe-het-zelvers
- Uiteindelijk gekocht door Bill Gates, die het integreerde voor de voorlopers van PC's (personal computers, '80)
- Opgevolgd door Visual Basic

STIJLEN VAN PROGRAMMEREN

IMPERATIEVE VERSUS DECLARATIEVE TALEN

IMPERATIEVE VERSUS DECLARATIEVE TALEN

Na FORTRAN en LISP volgden verschillende talen die zich lieten inspireren door een van beiden. Er ontstonden twee families van talen:

- Imperatief
  - denken zoals de computer: reeks opdrachten
  - nadruk ligt op algoritmisch denken
  - variabelen krijgen waarden in toekenningsopdrachten
  - FORTRAN, BASIC, C, C++, Java, Python, ...
- Declaratief:
  - denken zoals mensen/wiskundigen: WAT ipv HOE
  - sterke wiskundige basis
  - LISP, PROLOG, Haskell, ...

SUBGROEPEN BINNEN DECLARATIEVE TALEN

Binnen de groep van declaratieve kan je een aantal subgroepen onderscheiden:

- Functionele talen:
  - LISP, Scheme, Haskell, ML,…
  - gebaseerd op wiskundige functieleer
  - veel gebruikt voor AI
- Logische talen:
  - PROLOG voornaamste voorbeeld (PROgrammation en LOGique)
  - gebaseerd op wiskundige logica
  - veel gebruikt voor AI onderzoek (o.a. Watson en ISS …)
- Relationele vraagtalen:
  - SQL voornaamste voorbeeld
  - gebaseerd op relationele algebra
  - alom gebruikt bij relationele gegevensbanken
  - opgelet: vraagtalen zijn geen volledige programmeertalen

SUBGROEPEN BINNEN IMPERATIEVE TALEN

Ook imperatieve talen kunnen verder opgesplitst worden:

- Procedurale talen:
  - Programma's zijn een reeks instructies
  - Gebruiken procedures (=functies) om programma's te structureren
- Objectgerichte talen:
  - Programma's bestaan uit objecten die met elkaar communiceren
  - Gebruiken klassen met overerving om programma's te structureren

PROCEDURALE TALEN (1)

ALGOL 60 en ALGOL 68

- ALGOrithmic Language
- 1960: commissie van computerwetenschappers ontwerpt een “wetenschappelijk verantwoorde” (imperatieve) programmeertaal
- taal werd geen succes (te academisch, te ingewikkeld)
- 1968: vereenvoudigde versie: Algol 68

Pascal

- genoemd naar Blaise Pascal, 17de eeuwse wis- en natuurkundige, theoloog, bouwer van rekenmachine
- +/- 1970, Niklaus Wirth, Zwitserland
- sterk vereenvoudigde versie van Algol 60
- in de eerste plaats gericht op (verantwoorde) didactiek van gestructureerd programmeren
- groot succes van +/- 1980 tot +/- 1995

PROCEDURALE TALEN (2)

Ada

- genoemd naar Ada Lovelace, de eerste “programmeur” en vriendin van Charles Babbage
- +/- 1983, uit Pascal
- Amerikaanse defensie en ruimtevaart
- parallellisme en procescommunicatie

C

- opvolger van BCPL
- +/- 1975, ontwikkeld om Unix te (her)schrijven
- een taal enigszins tussen assembleertaal en hogere programmeertaal in
- tot +/- 1995 populairste taal onder professionele programmeurs, voor “informatietechnische” toepassingen

OBJECTGERICHTE TALEN

- tot +/- 1990 vrijwel uitsluitend experimenteel en academisch
- sinds 1995 wijd verbreid in de software-industrie (Vooral dankzij de opkomst van Java)
- Voorbeelden: Java, Smalltalk, Eiffel (voor didactiek), C (in beperkte maten), Dart (ontwikkeld door Google), Kotlin
- 'Upgrades' van bestaande talen: C++, Object Basic, Object Pascal
- Bij sommige talen hoorde ook een visuele ontwikkelomgeving (development environment). Bijvoorbeeld: Delphi voor Object Pascal, Visual Basic.NET voor Object Basic

SCRIPTTALEN

Scripttalen zijn een recenter fenomeen. Hun hoofddoel is het schrijven van scripts: kleine programma's die een bepaalde taak automatiseren. Ze hebben geen superstrikte definitie, maar vaak hebben ze volgende kenmerken:

- Zwak type systeem (bv. een lijst kan meerdere types data bevatten)
- Heel high level, met veel ingebouwde oplossingen voor vaak voorkomende opdrachten
- De taal wordt niet gecompileerd naar een lagere taal, maar lijn voor lijn geïnterpreteerd
- Meestal imperatief, ondersteunt vaak zowel proceduraal als objectgericht programmeren

Deze talen zijn populair voor hobbyprojecten en onderzoek, omdat ze toelaten snel en vlot code te schrijven.

Voorbeelden: Python, JavaScript, PHP.

TALEN VANDAAG

- Nieuwe programmeertalen kunnen redelijk snel gemaakt worden. Sommige mensen ontwerpen ze als hobby of voor onderzoek
- Nog steeds verschil tussen imperatief en declaratief, maar er is kruisbestuiving van features
- Nieuwe talen zijn bijna altijd geïnspireerd op oudere, maar verbeteren/veranderen bepaalde principes
- Soms worden talen ontwikkeld om een nieuw onstaan probleem op te lossen. Bijvoorbeeld: Kotlin en Swift zijn bedoeld voor efficiëntere app development op Android en iOS
- Heel af en toe zijn er talen met radicale aanpassingen. Bijvoorbeeld: Rust met zijn geheugenregels
- Oudere talen sterven langzaam af, maar sommigen blijven hardnekkig bestaan. Bijvoorbeeld: vrijwel alle systemen in de bankenwereld draaien nog steeds op COBOL. De reden is vaak dat het te duur is of te veel tijd kost om systemen te upgraden of te herschrijven in nieuwe talen

VAN HOGERE TALEN NAAR MACHINETALEN

COMPILERS EN INTERPRETERS

VAN HOGERE TALEN NAAR MACHINETALEN

We weten dat computers uiteindelijk instructies nodig hebben in machinetaal die door de computer begrepen kan worden. Hoe worden hogere talen getransformeerd naar de juiste machinetaal?

Er zijn voornamelijk twee technieken: **compilers en interpreters**. Talen kiezen zelf welke technieken ze gebruiken en hoe ze die implementeren.

COMPILERS VERSUS INTERPRETERS (1)

- Compilers (vertalers)
  - Lezen de volledige code in één keer
  - Genereren een resultaat in machinetaal
  - Voeren allerlei optimalisaties uit om de code sneller te laten werken
  - Voor zeer grote programma's kan compilatie enkele seconden tot zelfs minuten duren
  - Voorbeeld (min of meer): `.exe`-bestanden op Windows
- Interpreters (vertolkers):
  - Lezen code lijn per lijn
  - Geen compilatiestap → snellere opstart
  - Zijn veel langzamer tijdens uitvoering
  - Voorbeeld (min of meer): `.py`-bestanden uitvoeren met de Python interpreter

Compilers en interpreters zijn zelf programma's, vaak geschreven in lagere talen (assembly, C, ...). Ze hebben meerdere versies, om verschillende architecturen te ondersteunen.

COMPILERS VERSUS INTERPRETERS (2)

Moderne talen hebben complexe bouw- en optimalisatietechnieken waardoor het moeilijker is om een duidelijk onderscheid te maken tussen compilatie en interpretatie.

- Python wordt (meestal) “gecompileerd” naar “(Python) byte code” en die wordt dan uitgevoerd (meestal vertolkt) door een “Python virtuele machine”. Maar voor de programmeur “voelt” het als interpreteren
- Java wordt ook (meestal) “gecompileerd” naar “(Java) byte code” en die wordt dan uitgevoerd door een “Java virtuele machine”. Maar voor de programmeur voelt dit meer als compileren (want er verschijnen .class-bestanden in je mappen)