1 The anatomy of a computer

1.1 Computer programs

- Computers execute very basic instructions in rapid succession
- Computer program = sequence of instructions and decisions → tells computer what to do
- Hardware = physical computer devices
- Software = programs on the computer
- Programming = act of designing and implementing computer programs

1.2 The anatomy of a computer

- CPU = central possessing unit: performs program control and data processing
- Storage:
    - Primary storage = electronic chips (need electricity)
    - Secondary storage = hard disks (do not need electricity)
- Networks: by cable read data and programs from other devices

1.3 The Python programming language

- Python = a high-level-programming language (Guido van Rossum)
  →Portable, easy to learn and use, lots of packages
- Package: code for a particular problem domain

1.4 Programming environment

- Text editor = program for entering and modifying text (PyCharm)
- Case sensitive = uppercase and lowercase letters are not the same
- Python interpreter: reads program and executes the steps (run)
    - Compiler translates python code (source code) to byte code

1.5 Analyzing your first program

- Comment: provides information to the programmer
- Function = collection of instructions that perform a particular task
- → called by specifying the function name and its arguments
- String = a sequence of characters enclosed in a pair of quotation marks

1.6 Errors

- Compile-time error= syntax error: violation of the programming language rules that is detected when the code is translated into executable form (vb. Haakje niet sluiten)
- Run-time error= logic error: occurs when the program runs, produces unexpected results
- Exception= instruction is syntactically correct but impossible to perform (vb. Delen door 0)

1.7 Algorithm design

- Algorithm = a sequence of steps that is clear, executable, terminating
- Pseudocode = informal description of a sequence of steps for solving a problem

2 Programming with numbers and strings

2.1 Variables

2.1.1 Defining variables

- Variables: store values = storage location with a name
- Assignment statement: place a value in a variable
- Give the variable a new value, the old one gets replaced
- You can't change the variable with a value of a different type
- You can't use reserved words like class, if, and, else,…
- Use variable names in uppercase for variables that are constant, not being changed during the program

2.1.2 Number Types

Data Type of a value: specifies how the value is sorted in the computer and what operations can be performed on it

1. Primitive data type= provided by the language itself
    a. Numbers, strings, containers,…
2. User-defined data type = defined by user

Types of numbers:

1. Integer= without fractional part (geheel getal)
    a. Int(…)
2. Float(ing point numbers)= komma getal
    a. Float(…)
3. Number literal= getal uitgeschreven: komma → float anders integer
4. String= tekst van getal maken

2.2 Arithmetic

2.2.1 Arithmetic operations

- → expressions: combinations of variables, operators,…
- Vermenigvuldigen: *
- Delen: /

2.2.2 Powers

- Macht: **

2.2.3 Floor division and remainder

- Floor division= uitkomst zonder de cijfers na de komma EN niet afgerond naar boven
    - 13 // 6 = 2
- Modulus = remainder = rest van een deling
    - 13 % 6 = 1

2.2.4 Calling functions

- Absolute waarde: abs(…)

- Afronding: round(…)
- Maximum: max(X_1, X_1,…,X_n)
- minimum: min(X_1, X_1,…,X_n)

2.2.5 Mathematical functions

- Standard library: provides functions and data types for your code
- Library function had to be imported before it can be used:
- From *module* import *function*
- Import the entire module:
    - From module import *

- from math import
    - = built-in functions = defined as part of the language itself and used directly in programs
- Examples:
    - Vierkantswordtel: sqrt(…)
    - Float naar integer: trunc(…)
    - Radialen naar graden: degrees(…)
    - Graden naar radialen: radians(…)
    - Logaritme bepaalde base: log(…, base)
    - Exponentieel: exp(…)
- Counting floats → round-off errors!
    - 1/3= 0.33
    - 0.33 * 3= 0.99 != 1

2.3 Problem solving by hand

- Not immediately do it in python syntax, first write down how you're going to solve it
- Flowchart = mindmap

2.4 Strings

2.4.1 The string type

String = a sequence of characters

- String literal: duidt bepaalde string aan
    - Message= "Hello world"
- Compute the lengts of a string: len function
    - Length_message= len("World")
- Empty string= string with length 0

2.4.2 Concentration and repetition

Concentrate= meerdere strings samenvoegen tot 1

Vb. naam = voornaam + " " + achternaam

Repetition: in i string een bepaalde string een aantal keer herhalen

Vb. kusjes = x * 5 = XXXXX

2.4.3 Converting between numbers and strings

- Number to string: str(…)
- String to integer: int(…)
- String to float: float(…)

2.4.4 Strings and characters

- Strings are sequences of Unicode: access a character by its position
- Index of a character= position of the character in the string
- !!! Counting positions starts with position zero
  Vb. Naam = Kirsten
  - Eerste letter = Naam[0]
  - Laatste letter = Naam[6]

2.4.5 String methods

- Object = software entity that represents a value with certain behavior
- Method = collection of instructions that perform a particular task
- Examples:
  - Kleine letters: Naam.lower
  - Hoofdletters: Naam.upper
  - String veranderen: Naam2 = Naam.replace(“Kirsten”, “Gielen”)
  - Code character: ord(…)
    - Vb. ord(“H”)= 72
  - Character from a code: char(…)
    - Vb. char(97)= A
  - Escape sequence:
    - “ printen: \”
    - \ printen: \\
    - Enter: \n (binnen aanhalingstekens)
      - Vb. print(“K\nB\n”)
      - K
      - B

2.5 Input and output

2.5.3 Formatted output (p51)

- **f gebruiken bij een float, d gebruiken bij een integer, s bij een string**
- Afronden 5 cijfers na de komma: “%.5f” % getal
- Bepaald aantal plaatsen voorzien voor getal: “%10f” % getal
- Lege voorziene plaatsen vullen met 0: “%010f”
- Procent toevoegen: “%...%%”
- String format operator= specificeren hoe variabelen moeten voorkomen in de output (door middel van format specifiers)

2.6 Graphics: simple drawings

Module= from ezgraphics import …

2.6.1 Creating a window

- Graphics window is used for creating graphical drawings

- Geometric shapes are drawn on a canvas that is contained in a graphics window
- From ezgraphics import GraphicsWindow
- Win = graphicswindow(width,height)
    - graphicswindow(500.500)
- canvas = win.canvas
    - access the canvas contained in the graphics window
- win.wait()
    - waits for the user to close the window

3.6.2 Lines and Polygons

- The canvas has methods for drawing lines, rectangles and other shapes.
- Call the draw methods:
    - Canvas.drawLine(x1,y1,x2,y2)
    - Canvas.drawRect(x1,y1,width,height)
- ! origin is in the left top corner, y-axis is downwards and x-axis is to the right

3.6.3 Filled shapes and color

- The canvas stores the current drawing parameters used to draw shapes and text
- Colors can be specified by name or by their red, green and blue components.
    - Canvas.setOutline(colorName)
    - Canvas.drawRect(x1,y1,width,height)

3.6.4 Ovals, Circles and text

- Canvas.drawOval(x,y,width,height)
- Canvas.drawText(x1,y1,"TEKST")
    - Anchor point is in the left corner of the beginning of the text

3 Decisions

3.1 The if statement

- Allows a program to carry out different actions depending on the data to be pocessed
- Compound statements= multiple lines consisting of a header and a statement block

```
Vb. If getal < 4:                    #header
        Getal = volume + 2           #statement block
```

3.2 Relational operators

- Examples:
    - Groter dan of gelijk aan: >=
    - Teller = teller + 1 → teller += 1
    - Teller = teller* 2 → teller *= 2
    - Niet gelijk aan: !=
    - Gelijk aan: ==
- Alfabetisch vooraan: string1 < string2
- Gelijke woorden: string1 == string2
- !!! hoofdletters voor elke gewone letter, getallen voor letters, spatie voor letters

3.3 Nested branches

- Nested set op statements= statements in another statement → multiple levels
- Hand-Tracing: simulate the program on a piece of paper in pseudocode/python code

3.5 Flowcharts

- Flowcharts= made up of alaments for tasksn input, output, decisions
- Each branch can contain a task of further decision

3.7 Boolean variables and operators

- Bool data type: two values: True or False
- Boolean operators: And, Or, Not
- Short-Circuit evaluation: and and or operators
    - Evaluated from left to right, stops when truth value is determined
    - Fastest path for computing the result of a Boolean expression

3.8 Analyzing strings (p108)

- Substring in a string: Kir in Kirsten → True/False
- Niet gemeenschappelijke letters: Kirsten.count(kir)
- Substring end of string: Kirsten.endswith(kir)
- Substring begin of string: Kirsten.beginswith(Kir)
- Beginindex substring in string: Kirsten.fine(kir)
- Alleen kleine letters: Kirsten.islower()
- Alleen letters alfabet: Kirsten.isalpha()

4 Loops

4.1 The while loop

- While loop: executes instructions repeatedly while a condition is true
- Infifint loop: condition never becomes false
- Event controlled loop:
    - Don't know how many times the loop will work
    - Indefinite loop
    - Event changes the loop expression from true to false
- Counter controlled loop:
    - Do know how many times the loop will work
    - Definite loop
- Off-by-one error: common error when programming loops, think through simple test cases to avoid this type of error.
- Named argument: end=""
    - Print("00", end="")
    - Print(3+4)
    - 007

4.2 Problem solving: Hand-Tracing

- Hand-tracing: simulation of code execution in which you step through instructions and track the values of the variables.
- Helps understand how an unfamiliar algorithm works
- Shows errors in your code

4.3 Application: Processing sentinel values

- Sentinel= ingegeven waar de stopt het ingeven van meerdere waarden → programa kan beginnen met de loop/berekening
- Storyboard: consists of annotated sketches for each step in an action sequence.

4.6 The for loop

- For letter in name
    - Name = container
- For I in range(6) → 0, 1, 2, 3, 4, 5
- For i in range(2.5) → 2, 3, 4
- For I in range(1,15,2) → 1, 3, 5, 7, 9,…
- For I in range (10, 0,-2) → 10, 8, 6, 4, 2

4.7 Nested loops

- Loop inside another loop statement

6 Lists

6.1 Basic properties of lists

- List = container that stores a sequence of values
- List = [5, 7, 6, 45, 25 ,1 ,5]
- Empty_list = []
- Waarde op bepaalde plaats: List[5]
- Out-of-range error= bounds error= waarde op bepaalde plaats vragen waarvan de plaats niet in de lijst zit
- List reference: specifies the location of a list, copying the reference yields a second reference to the same list.

6.2 List operations

- Toevoegen op bepaalde plaats: namen.insert(2, “Kirsten”)
- Index waarde in lijst: namen.index(“Kirsten”)
- Verwijderen uit lijst: namen.remove(“Kirsten”)
- Verwijderen op bepaalde positie: namen.pop(4)
- Laatste waarde verwijderen: namen.pop()
- Klein naar groot: waarden.sort(5, 3, 2, 1) = [1, 2, 3, 5]
- Sublist van een lijst: namen[2 : 5]

6.4 Using lists with functions

- Tuple= niet aanpasbare lijst

6.7 Tables

- Tabel = [[1,2,3], [5, 2, 6], [8, 6, 1]]
- Bepaald element op plaats: lijst [1] [4]

Extra: correctheidsbewijzen

➔ Formeel en wiskundig bewijzen van je programma
➔ Algoritme met while-lus

Template algoritme:

…

<initialisatie>

While <lus-voorwaarde>

    <lus-opdrachten>

…

Basisstappenplan:

- **<u>Gegeven</u>**
    - Algortime
STAP 1 ○ Pre-conditie ‘beginstoestand’
STAP 2 ○ Post-conditie ‘eindtoestand’

- **<u>Te bewijzen</u>: algoritme is correct, d.w.z**.

STAP 3 ○ Vertrekkende van de preconditie zal, door het behoud van de **<u>lus-invariant</u>** gegarandeerd waar zijn: postconditie

STAP 4 ○ En de lus zál eindigen eindigheid

Stap 1: Preconditie

= beginvoorwaarde van het algoritme

Algoritme mag ervan uitgaan dat deze voorwaarden waar zijn in het begin van de uitvoering.

Vertelt iets over het type en de waarden van variabelen.

…

PRECONDITIE:

 # PRE: n is een natuurlijk getal

<initialisatie>

While <lus-voorwaarde>

<lus-opdrachten>

…

Stap 2: Postconditie

= resultaat van het algoritme

Deze voorwaarden moeten waar zijn aan het einde van de uitvoering.

Vertelt het resultaat van het algoritme.

…

PRECONDITIE:

 # PRE: n is een natuurlijk getal

<initialisatie>

While <lus-voorwaarde>

<lus-opdrachten>

POSTCONDITIE:

 # POST: som bevat de waarde van 1 + 2 + 3 + … + n

…

Stap 3: Partiële correctheid aan de hand van de lus-invariant

	Som	**Teller**	**Wat weet je?**	=
0	0	0	Som = 0	0 <= n
Na 1 initialisaties	1	1	Som = 1	1 <= n
Na 2 initialisaties	3	2	Som = 1 +2	2 <= n
Na 3 initialisaties	6	3	Som = 1 + 2 + 3	3 <= n
Na k initialisaties	?	k	Som = 1 + 2 + … + k	K <= n

voorwaarde die waar zijn elke keer dat de lus-opdrachten zijn uitgevoerd

= afhankelijk van lus-variabelen (variabelen gebruikt in de lus-voorwaarden/lus-opdrachten)

Stap 3.1: Specificatie

…

PRECONDITIE:

 # PRE: n is een natuurlijk getal

<initialisatie>

While <lus-voorwaarde>

    <lus-opdrachten>

INVARIANT:

#INVARIANT: som bevat 1 + 2 + … + teller, teller <= n

POSTCONDITIE:

 # POST: som bevat de waarde van 1 + 2 + 3 + … + n

…

Stap 3.2: lus-invariant – bewijs geldigheid na initialisatie

(Lus-)invariant moet gelden aan het begin van de lus.

…

PRECONDITIE:

 # PRE: n is een natuurlijk getal

<initialisatie>

#INVARIANT: som bevat 1 + 2 + … + teller, teller <= n → OK

While <lus-voorwaarde>

    <lus-opdrachten>

INVARIANT:

#INVARIANT: som bevat 1 + 2 + … + teller, teller <= n

POSTCONDITIE:

 # POST: som bevat de waarde van 1 + 2 + 3 + … + n

…

Stap 3.3 lus-invariant – bewijs behoud na elke iteratie

(Lus-)invariant moet gelden aan het begin van elke iteratie, behouden na elke iteratie.

…

PRECONDITIE:

 # PRE: n is een natuurlijk getal

<initialisatie>

#INVARIANT: som bevat 1 + 2 + … + teller, teller <= n → OK

While <lus-voorwaarde>

    # ?INVARIANT?

    <lus-opdracht>

    # ?INVARIANT?

    <lus-opdracht>

    # ?INVARIANT?

INVARIANT:

#INVARIANT: som bevat 1 + 2 + … + teller, teller <= n

POSTCONDITIE:

 # POST: som bevat de waarde van 1 + 2 + 3 + … + n

…

Stap 3.4: Lus-invariant + stop-voorwaarde lus → post-conditie

…

PRECONDITIE:

 # PRE: n is een natuurlijk getal

<initialisatie>

#INVARIANT: som bevat 1 + 2 + … + teller, teller <= n → OK

While <lus-voorwaarde>

    # ?INVARIANT?

    <lus-opdracht>

    # ?INVARIANT?

    <lus-opdracht>

    # ?INVARIANT?

→INVARIANT:

#INVARIANT: som bevat 1 + 2 + … + teller, teller <= n

→**POSTCONDITIE:**

 # POST: som bevat de waarde van 1 + 2 + 3 + … + n

…

Stap 4: eindigheid

= aantonen bij elke iteratie het einde van de lus dichterbij komt + geraakt met eindig aantal stappen

➔ Vindt een variant
- Wiskundige uitdrukking in termen van variabelen
- Strikte ondergrens
- Daalt strikt monotoon bij elke iteratie
- Aantal stappen is eindig

Stap 4.1: specificatie van de variant

Herinner de voorwaarden: uitdrukking, begrensd,…

Variant: n – teller

Stap 4.2: bewijs van strikte ondergrens van de variant

Door middel van andere variabelen en voorwaarden ondergrens afleiden uit variant

Stap 4.3: bewijs van strikt monotone daling

Stap 4.4: bewijs eindig aantal stappen

5 Functions

5.1 functions as Black Boxes

- Function = named sequence of instructions
- Arguments = input function = supplied when a function is called
- Return value = output function = result function computes
- Returning a value from a function is not the same as producing program output
  - Return value: only returned to the point where the function is called
  - Program output: printed result

5.2 Implementing and testing functions

- Parameter variables: variable that is used in the arguments of your function
- Header of the function:
  - Def function_name(parameter variable)
- body of the function: everything after the header
- function comments: explain purpose function, meaning parameter variables, meaning return value, special requirements

5.3 Parameter passing

- parameter variables= formal parameters: hold the arguments (values) supplied in the function call

5.4 Return statement

- return statement: terminates a function call and yields the function result
- compound statement= header on one line and the body on the other
    - single line-form: single statement

5.5 Functions without return values

- some functions may not return a value, but they can produce an output

5.7 Stepwise refinement

- decompose complex tasks into simpler ones
- use functions!
- step by step
- waarde- en referentiesemantiek:
    - als een variabele een bepaalde waarde heeft en men creëert een nieuwe variabele die men gelijk stelt aan die variabele dan neemt de nieuwe variabele de waarde over, hij wordt gekopieerd
    - bij lijsten kopieert met de verwijzing naar de lijst ipv de lijst zelf. Als men in de nieuwe variabele dan dingen aanpast in de lijst, dan zijn die ook aangepast in de originele lijst
    - waarde-semantiek: boolean
    - referentiesemantiek: lijst, tabel, ...
- stubs= function that returns a simple value that is sufficient for testing another function

5.8 Variable scope

- scope of a variable = the part of the program in which it is visible, in wich you can access it
- local variable= defined within a function or code block
    - only available at that point till the end of the block/function
- global variable= defined outside of a function
    - visible to all functions