SIT22001 **PROGRAMMING** Ⅰ
Lecture 1

Fall 2019

School of Global Entrepreneurship & ICT
Handong Global University

OUTLINE

Goals of the course

What is computation ?

Computational thinking

About Python

2D robot control

Reading assignment:

- Chapter 1 of the textbook
- Learning programming with robots
  (You may download the **pdf** file from our **Hisnet site**)

GOALS OF THE COURSE

Two-level goals

- Building up a basis on ICT (Information and Communications Technology)
- **Computational thinking and programming** (but not learning a programming language **Python**)

Think like a computer scientist for problem solving!

WHAT IS COMPUTATION ?

Problem solving with a computer

1. Finding facts(conditions) that a solution satisfies
2. **Designing an algorithm(recipe) to find a solution**
3. **Mapping the algorithm to a program**
4. Understanding abilities and limitations of your program

"Algorithm" is at the heart!

Knowledge

Declarative

statement of facts

$\sqrt{x}$ is $\pm$y such that y^2 is x.

Imperative

recipes for deducing information
"how to" knowledge

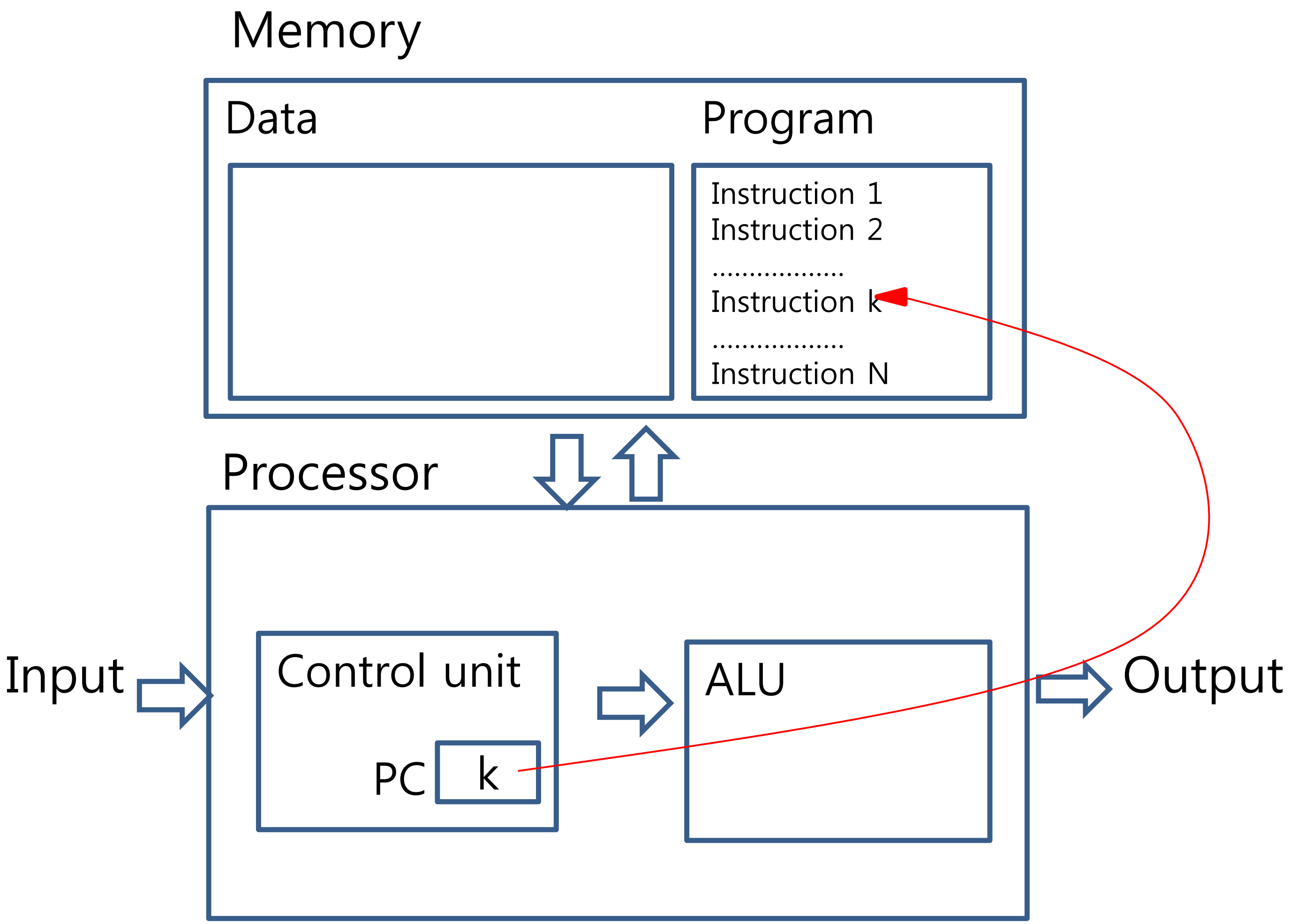
Start with guess G.
If $G^2 \approx x$, stop and return $\pm G$.
Otherwise, G ← (G + x/G)/2.
Repeat.

Heron of Alexandria(10-70 AD)
Ancient Babylonians

Fixed program computers

Atanasoff and Berry(1941): a linear equation solver
Alan Turing: bombe machine
Calculators

Stored program computers (Von Neumann machine)



Summary

Computation is **solving** a problem with a **program**.

A **program** is a **realization** of an **algorithm**(recipe) on a **computer**.

An **algorithm** is a **finte sequence** of **instructions** to do a task.
imperative knowledge
(for humans)

An **algorithm** should be **refined** enough to be **easily translated** into a **program** using a program language.
(for computers)

How to design an algorithm : **top-down design**

How to convert it to a program: **coding** and **debugging**

What to do with **computers(programs)** ?

Top-down design

Decomposing a problem into smaller sub-problems

Decomposing each of the smaller sub-problems recursively until every sub-problem is simple enough to map to a few instructions in a program language

Divide and conquer
Multi-level abstraction

Coding and debugging

Coding is "a process of fighting with bugs (errors)."

Syntax error: Python cannot understand your program, and refuses to execute it.
Runtime error: At runtime, your program suddenly terminates with an error message.
Semantic error: Your program runs without error messages, but does not do what it is supposed to do.

Why making **such bugs (errors) ?**
Well, … , that is the **difference** between **humans** and **computers**.

What to do with computers?

According to **Turing-Church Thesis,** modern computers are essentially equivalent to a **Turing machine(stored program computer).**

What kind of problems can we solve with a Turing machine?

Decidable problems

- **Tractable problems : good algorithms**
- Intractable problems: no good algorithms
  - e.g., travelling salesman's problem
  - **approximate algorithms**

Undecidable problems: no algorithms ever found

- e.g. halting problem

ABOUT PYTHON

General vs Targeted

Low vs **High**

Compiled vs **Interpreted**

Python is relative **young** but one of the most **popular** programming languages

Open software

Why Python ?

A programming language easy to learn and very powerful

- Used in many universities for introductory courses
- A main language used for web programming at Google
- Widely used in scientific computation, e.g., at NASA
- Large portions of games written in Python (Civilization IV)

Once you learnt programming in one language, it is relatively easy to learn another language, such as C++ or Java.

Characteristics of Python

Instruction set

Arithmetic and logical operations
    +, -, *, /, and **
    and, or, not
Assignment
Conditionals
Iterations
Input/output

for defining expressions

No pointers

No explicit declarations

Why programming ?

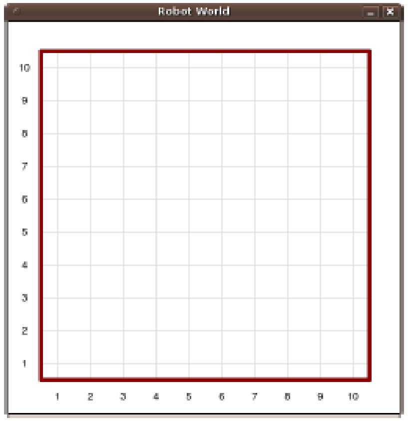
Every scientist and **engineer must know** some programming. It is part of basic university education, like calculus, linear algebra, introductory physics and chemistry, or English.

Alan Perlis 1961

After half a century later, we should change it as follows:

Every student in a **university should learn** some programming. It is part of basic education, like calculus, linear algebra, introductory physics and chemistry, or English.

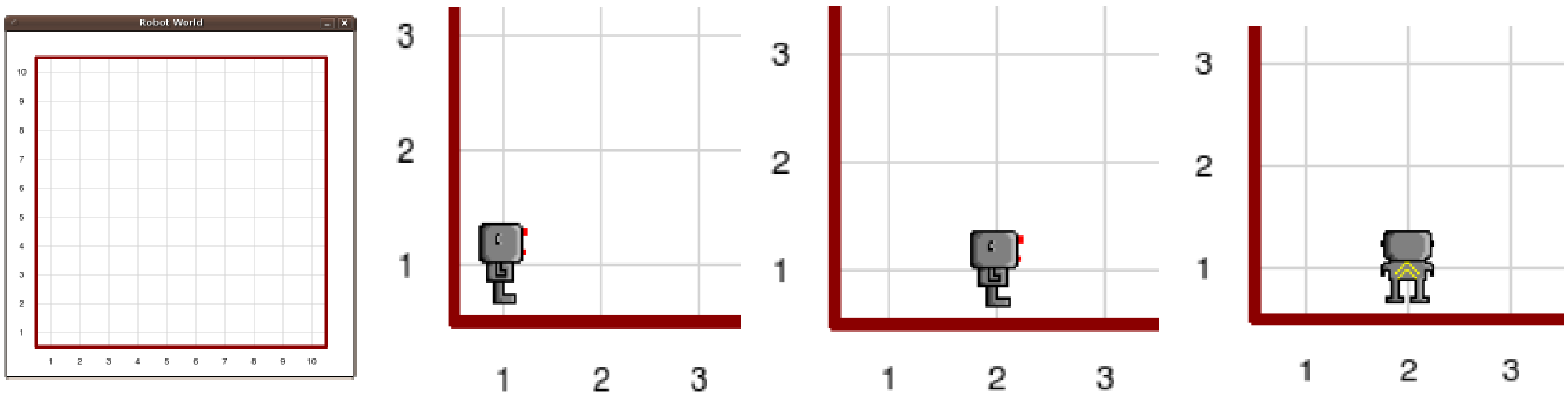
2D ROBOT CONTROL



- A small grid-like 2D world
- Basic actions
  - move (): moving one grid forward
  - turn_left (): turning left by 90°
  - pick_beeper(): picking up beepers
  - drop_beeper(): putting down beepers
- **Our own instructions: functions**
- Comments

- Interactive mode
- Script mode: Python programs (scripts)

Interactive mode



```
>>>from cs1robots import *
>>>create_world()
>>>hubo = Robot()
>>>hubo.move()
>>>hubo.turn_left()
```

Script mode

```
from cs1robots import *
create_world()
hubo = Robot()
hubo.move()
hubo.turn_left()
```

Functions

A **function definition** specifies the **name** of a function and its **body** that consists of the sequence of statements that are executed when the function is called.

```
def print_message():
   print ("SIT22001 is fantastic!")
   print ("Programming is fun!")
```

You can call a function inside another function:

```
def repeat_message():
   print_message()
   print_message()
```

Flow of execution

```
def print_message():
    print (“SIT22001 is fantastic!")
    print (“Programming is so much fun")

def repeat_message():
    print_message()
    print_message()

repeat_message()
print (“Done”)
```

function definitions

function calls

Execution begins at the first statement. Statements are executed **one by one, top to bottom.**

Function definitions do not change the flow of execution but only define a function.

Function calls are like detours in the flow of execution.

Comments

```
# create a robot with one beeper
hubo = Robot(beepers = 1)

# move one step forward
hubo.move()

# turn left 90 degrees
hubo.turn_left()
```

dot notation

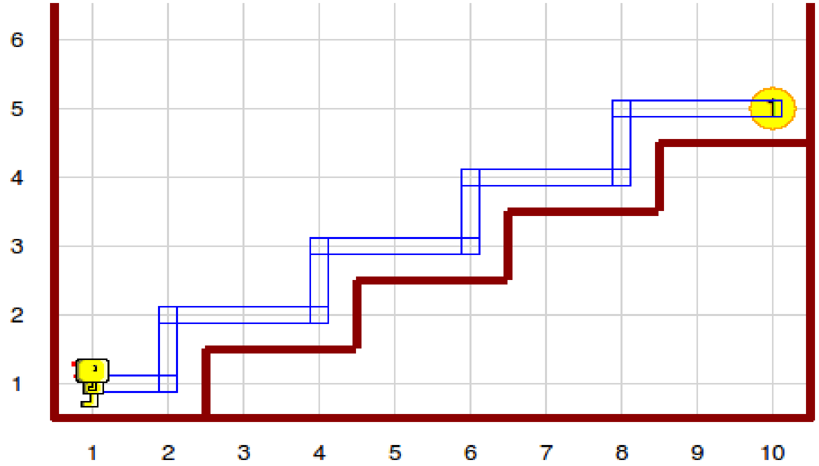
Turning right

Define a function!

```
def turn_right():
   hubo.turn_left()
   hubo.turn_left()
   hubo.turn_left()
```

Newspaper delivery

Hubo should climb the stairs to the front door, drop a newspaper there, and return to his starting point.



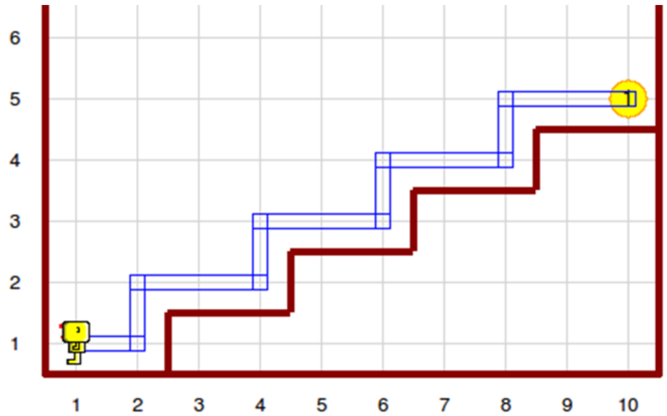
Algorithm(pseudo code):

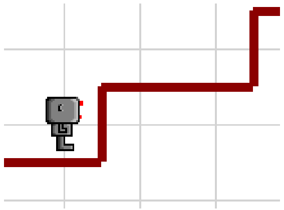
Move to the stairs
Climb up four stairs
Drop the newspaper
Turn around
Climb down four stairs
Move back to the origin

Python version:

```
hubo.move()
climb_up_four_stairs()
hubo.drop_beeper()
turn_around()
climb_down_four_stairs()
hubo.move()
```

Climbing up stairs





```
def climb_up_four_stairs():
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()

def climb_up_one_stair():
    hubo.turn_left()
    hubo.move()
    turn_right()
    hubo.move()
    hubo.move()

def turn_around():
    hubo.turn_left()
    hubo.turn_left()
```

Iteration: for-loops

We should **avoid writing** the same code **repeatedly.**
A `for`-loop allows us to write it more elegantly:

```
def climb_up_four_stairs():
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()

def climb_up_four_stairs():
    for i in range(4):
        climb_up_one_stair()
```

To repeat the same instruction 4 times:

```
for i in range(4):
    print ("SIT22001 is fantastic!")
```

for-loop

Don't forget the indentation!

What is the difference between the following two programs?

```
for i in range(4):
    print ("SIT22001 is great!")
    print ("I love programming!")
```

```
for i in range(4):
    print ("SIT22001 is great!")
print ("I love programming!")
```