

Programming for Scientists

COMP1730 & COMP6730

*We acknowledge and celebrate the First Australians
on whose traditional lands we meet,
and pay our respect to the elders past and present.*



Australian
National
University

Course conveners

<https://comp.anu.edu.au/courses/comp1730/people/>

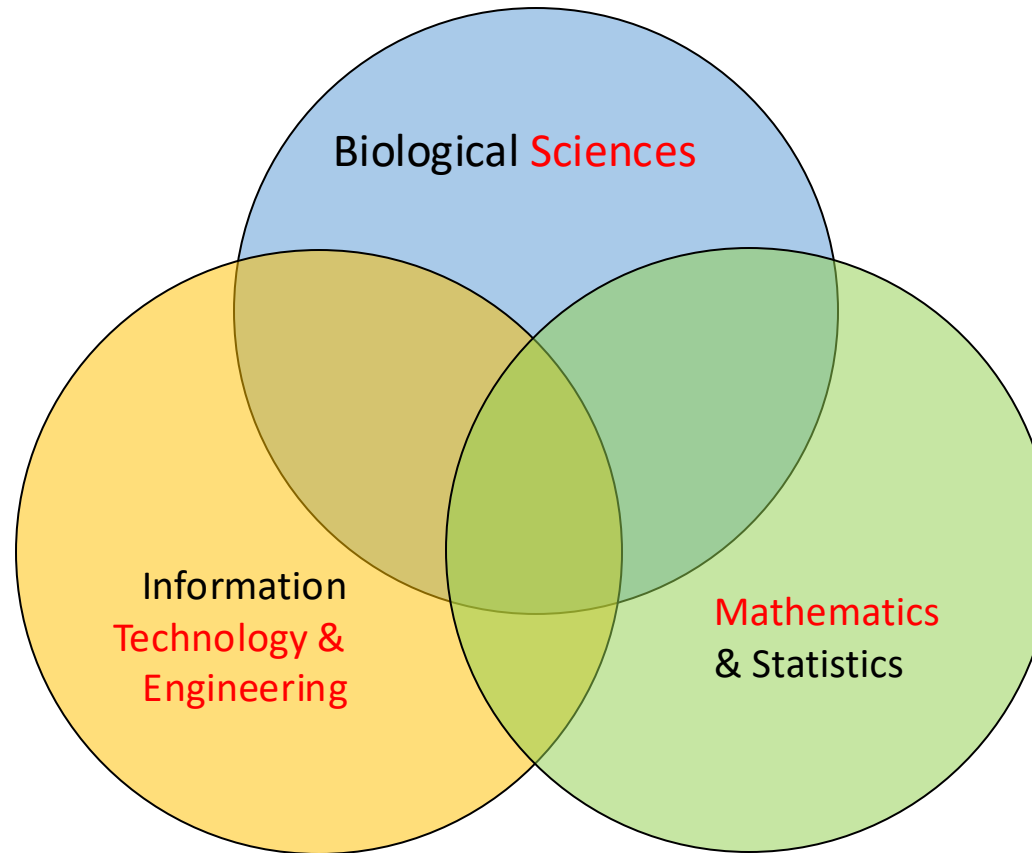


A/Prof. Minh Bui



Dr. Thang Bui

I work at the interface of STEM



Tutors (incomplete list)

<https://comp.anu.edu.au/courses/comp1730/people/>

Hancheng Shao



Alexei Khorev



E Wen Wong



Jeevika Shrivastava



Rohan Khatwani



Bodan Liu



Enze Tao



Nutthadech (K) Ban



Cathy Cheung



Yixiang Yin



Week 1 announcement

- Complete week 1 checklist on [course website](#)
- Lectures: Mondays 11am – 13pm
- Labs: only from week 2 (**no lab in week 1**)
 - Lab in week 2 is to practice concepts from week 1 lecture, and so on.
- Install Fest in week 1:
 - Friday, 25/7, 6-9pm, location to be confirmed
 - **Food provided!**

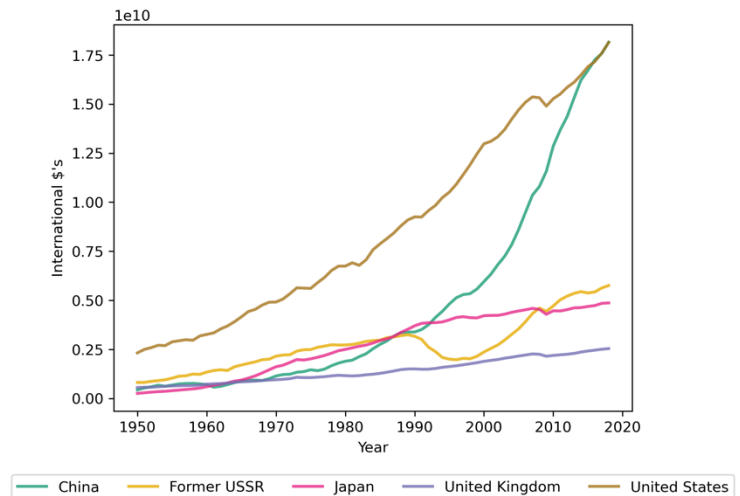
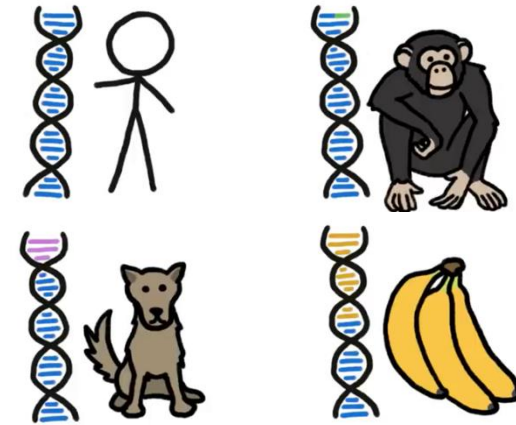
Lecture outline

- Why programming for scientists?
- Course overview
- Info, contacts & schedule
- Assessment scheme
- Academic integrity
- Student representatives

Why programming for scientists?

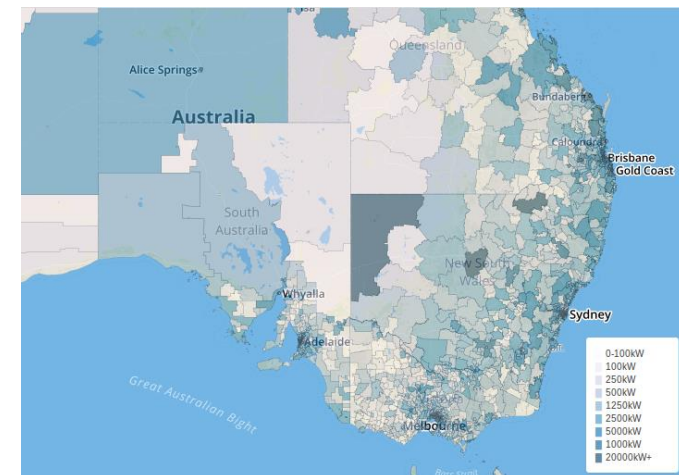
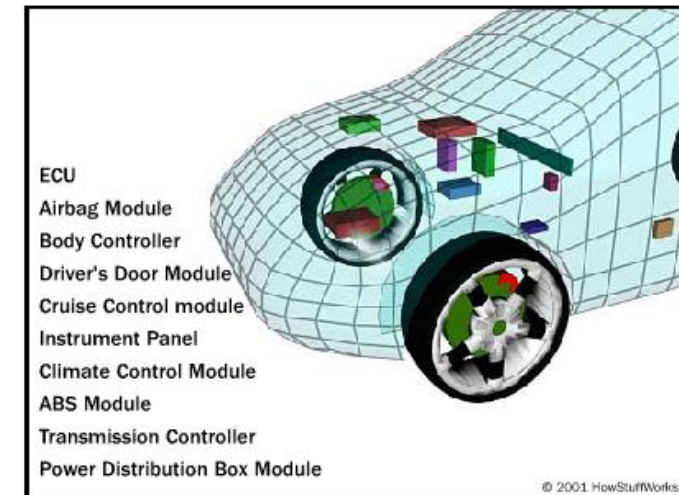
“Science rests on data, processing data needs software.”

- **Biology:** Use DNA data to understand the evolution of Life on Earth and track COVID-19 virus variants.
(<https://iqtree.github.io>)
- **Economics:** Modelling GDP growth over time and across countries.
(<https://quantecon.org>)



Why programming for scientists?

- Technical systems increasingly run on software
 - **Engineering:** Software on modern cars has >100M lines of code (not yet counting self-driving)
- **Optimisation** is needed to solve large-scale design challenges
 - Solar/Wind produced 28% of Australia's electricity in 2024. How do we design the grid to work with 100%?



This course

- Is a first course in programming
 - Focused in transferable, practical skills
 - Understand some basic CS concepts; build foundation for later courses
 - Coding languages come and they go – but the good coding practice is relevant to all languages
 - Useful to those in science and engineering.
- A beginners course – no prior experience required.
 - But this doesn't mean that it will be easy!

Expected outcomes

- As scientist and engineer, you need to understand how software works, and perhaps extend it:
 - understand algorithms and implementation to interpret and explain their results
 - debug programs (find and correct errors)
 - modify existing programs to solve your (unique) problem
- By the end of the course, we hope you'll tackle a novel problem by saying

Hey, I can just write a program to solve that...

Why Python?

- Python is nowadays the *most popular* programming language
 - particularly for science and engineering uses
 - We'll use Python 3 (not Python 2)
- Open source, available on most platforms
- Many modules, >200 Python standard libraries, >100,000 Python packages.
- This is not a course on programming in python; it's a course on programming, that uses python

Course info and contacts

- <https://comp.anu.edu.au/courses/comp1730/communication/>
- Read Announcements on Ed Discussion (<https://edstem.org/au/courses/25182/discussion>), esp. weekly information
- All materials (labs, assignments, quizzes, exam) on Ed Lessons: <https://edstem.org/au/courses/25182/lessons>
- To ask a question:
 - Ask your tutors in lab!
 - Use Ed Discussion
- If you have private questions:
 - Technical questions (e.g. Python code), tick “Private” on Ed Discussion post.
 - Personal/administrative questions: email comp1730@anu.edu.au from your ANU account (@anu.edu.au). Other emails will be ignored.
 - Or drop-in at Minh’s office hours: Mondays 1-2pm

Three simple rules when asking questions

- Read before you post
 - Before posting a question, check if your question has already been asked/answered
- Give your post a **good, descriptive** topic
 - Don't write *A question* or something like that
 - Good example: *Variable assignment: why does the value not change?*
- You **must not post** solutions to assignment problems
 - This applies to any forums

Demographic survey

Your degree?

A	Computer Science	12%
B	Engineering	25%
C	Science	21%
D	Business/Economics	28%
E	Arts or Social Science	3%
F	Law	0%
G	Medicine	3%
H	Other/Double Degree/not in a degree program	12%

Your experience?

A	No previous programming experience	35%
B	Have only used it following detailed instructions (e.g., in a tutorial or lab)	31%
C	Know the basics; can write a small/simple program from scratch	28%
D	Know the language well; have written or modified several larger programs	3%
E	Know the language very well; familiar with modern software design, documentation, and testing practices	3%

Only 62 responses so far, please fill out the survey on EdStem this week!

Demographic survey

Which platform you are using?

A	Mac laptop or desktop	50%
B	Windows laptop or desktop	62%
C	Unix, GNU/Linux, Ubuntu	18%
D	iPad or Android pad	28%
E	Not sure	0%
F	I don't usually use a computer	0%

Get yourself familiar with Linux as mid-term test and final exam
will be conducted on Linux lab computers!

Course structure – 1st half of semester

- Week 1 – Basics and functional abstraction
- Week 2 – Data types, expressions, and functions
- Week 3 – Code branching and Iteration
- Week 4 – Sequence data & code quality
- Week 5 – Strings & slicing, testing & debugging
- Week 6 – Lists, Data analysis & visualisation

Teaching break

Live recording on echo360 available

See: <https://comp.anu.edu.au/courses/comp1730/lectures/>

Labs/tutorials (from week 2) and drop-ins

- You need to choose one lab (90 min each) on MyTimeTable:
 - Mon 13:00-14:30, 15:00-16:30
 - Tue 11:00-12:30, 13:00-14:30
 - Wed 9:00-10:30, 10:00-11:30, 14:00-15:30, 15:00-16:30
 - Thu 11:00-12:30, 16:00-17:30
 - Fri 11:00-12:30
 - We may open another lab on Tue 13:00-14:30 if capacity exceeded.
 - Lab exercises are not meant to solve within 2 hours. You are expected to spend another 6 hours to study the course.
- Drop-ins:
 - 30 min immediately after each lab
 - Open to all students, not just those in that lab.

Assessment scheme

Component	Weight	Hurdle	Release date	Due date/Exam date
Homework 1	3%	No	28/07/2025 (Wk 2 Mon)	03/08/2025, 23:55PM (Wk 2 Sun)
Homework 2	3%	No	04/08/2025 (Wk 3 Mon)	10/08/2025, 23:55PM (Wk 3 Sun)
Homework 3	4%	No	11/08/2025 (Wk 4 Mon)	24/08/2025, 23:55PM (Wk 5 Sun)
Mid-term Test	20%	7%/20%		Week 6
Homework 4	5%	No	15/09/2025 (Wk 7 Mon)	28/09/2025, 23:55PM (Wk 8 Sun)
Project assignment	25%	No	Week 8	12/10/2025, 23:55PM (Wk 10 Sun)
Final Exam	40%	10%/40%		Exam period

- Mid-term test is a **hurdle** – at least 7/20 to continue the course
- Final exam is also a **hurdle** – at least 10/40 to pass
- Project assignment interview in week 11 is **mandatory** – absence will lead to zero mark for it

<https://comp.anu.edu.au/courses/comp1730/assessments/>

Marking/grading

- If you fail the hurdle in mid-term test, you should **unenroll** from the course
- Final mark is the sum of all assessment marks, with 50% required to pass (**mind the mid-term and final exam hurdle!**).
- If you are close to higher grade (e.g. 48-49, 78-79), we may round it up based on your **lab participation**: lab attendance, Ed lessons completion, Ed Quizzes completion (see Assessment page).
- Any submitted work may be subject to an **extra oral assessment**, which can change the mark.
- All assessment **deadlines are hard** – late submissions without an approved extension won't be accepted
 - Unlike other course, there is **no partial mark** for late submissions.

Academic integrity

- Academic integrity is taken seriously at ANU!
 - <https://www.legislation.gov.au/Details/F2021L00997/>
 - Being uninformed of or misunderstanding the Rule is never an excuse for a breach of academic integrity
 - Any student is expected to undertake the online Academic Integrity modules
- Discussing programming problems (e.g. from labs) and ways to solve them with other students is a great way to learn
 - just don't discuss assessment problems
- All assignments/exam are *individual*. **You must write your own code.**
 - Submitting solutions you found on the web or enlisting others (**including AI such as ChatGPT**) are all forms of **cheating**.
 - In intro course like this, **learning the basics without AI is important for you.**

Reading: Course Textbooks

Alex Downey (2016) *Think Python*, 2nd Edition

Sundnes (2020) Introduction to Scientific Programming with Python (ItSPwP)
(electronic copies available at: <https://comp.anu.edu.au/courses/comp1730/resources/>)

- Other good resources:

<https://docs.python.org/3/tutorial/index.html>

- Other books:

Al Sweigart (2015) *Automate the Boring Stuff with Python*

Bill Lubanovic (2019) *Introducing Python*, 2nd Edition

- When reading other python books, make sure they are python 3!
- Be careful with web resources – some are great (eg. docs.python.org). Many aren't.

- CSS
- CLASS REPRESENTATIVES

Class Student Representation is an important component of the teaching and learning quality assurance and quality improvement processes within the ANU College of Systems and Society (CSS).

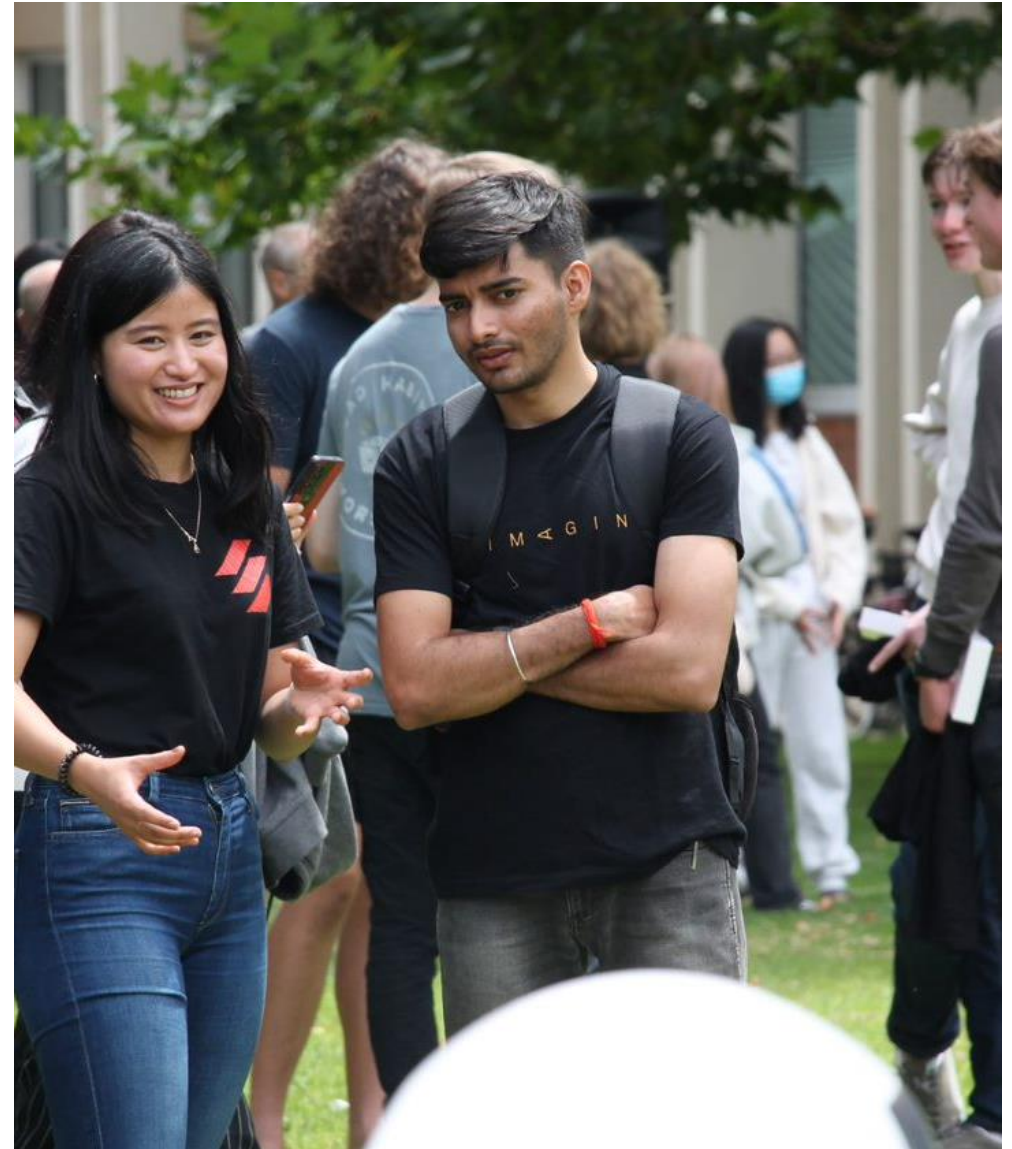
Each semester, we put out a call for Class Representatives for all ANU College of Systems and Society (CSS) courses. Students can nominate themselves for one or more of the courses they are enrolled in.



Australian
National
University

• Roles and responsibilities:

- The role of Student Representatives is to provide ongoing constructive feedback on behalf of the student cohort to Course Conveners and to Associate Directors (Education) for continuous improvements to the course.
- Act as the official liaison between your peers and convener.
- Be available and proactive in gathering feedback from your classmates.
- Provide reports on course feedback to your course convener.
- Close the feedback loop by reporting back to the class the outcomes of your feedback.
- Note: Class representatives will need to be comfortable with their contact details being made available via Wattle to all students in the class.
- For more information regarding roles and responsibilities, contact:
- ANUSA CSS representatives (sa.cecc@anu.edu.au).





• Why become a class representative?

- **Ensure students have a voice** to their course convener, lecturer, tutors, and College.
- **Develop skills sought by employers**, including interpersonal, dispute resolution, leadership and communication skills.
- **Become empowered.** Play an active role in determining the direction of your education.
- **Become more aware of issues influencing your University** and current issues in higher education.
- **Course design and delivery.** Help shape the delivery of your current courses, as well as future improvements for following years.

**Want to be a class representative?
Nominate today!**

Please nominate yourself to your course convener by end of Week 2

Programming example

- You want to calculate the monthly repayment of a \$500,000 home loan:
 - Use an online mortgage calculator?
- For all loan terms in 20-30 years and an interest rate of 2% to 6%.
 - You have to repeat using calculator many times
- The formula is:

$$A = P \frac{r(1 + r)^n}{(1 + r)^n - 1}$$

where P is principal (loan amount), r is interest rate, n is loan duration

- Let's write a program!

Try now on Ed Lessons

This is the first way:

```
▶ Run PYTHON   
1 P = 500000  
2 r = 0.06  
3 n = 30  
4 year_repayment = P * r * (1+r)**n / ((1+r)**n - 1)  
5 print(year_repayment/12) 
```

This is the second way:

```
▶ Run PYTHON   
1 P = 500000  
2 r = 0.06/12  
3 n = 30*12  
4 month_repayment = P * r * (1+r)**n / ((1+r)**n - 1)  
5 print(month_repayment) 
```

QUESTIONS:

- Run both codes, do you notice that they return different values?
- Which one do you think is the correct way?
- Can it be that both codes are incorrect?
- Is there any other alternative way of computing the repayments?

Part 2: Functional abstraction



Australian
National
University

Outline

- Meet Karel the robot
- Libraries, modules, namespaces
- Functional abstraction & decomposition
- The Python language: first steps

History behind Karel the robot

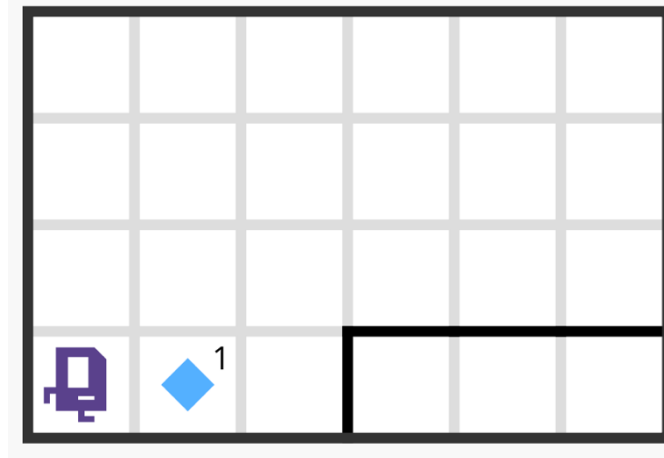
- Gentle *introductory programming environment* by Richard Pattis (Stanford Uni) in the 1970s
- Well-received by millions of students worldwide
- You will *teach* a robot to solve simple problems
- Robot is named after playwright Karel Capek, who introduced the word “robot” to English in 1920 play [Rossum's Universal Robots](#).

What is Karel?

- Very simple robot living in a very simple world
- Using commands (instructions), we can direct Karel to perform certain tasks within its world
- Process of specifying those commands is called *programming*
- Initially, Karel understands a small set of predefined commands, but a **key part of programming is to define new commands that extend its initial capabilities**
- “**Karel language**” is much simpler than Python

Karel's world

- Karel lives in a rectangular grid of **columns** and **rows**
- Example of a world with 6 columns and 4 rows:



- The world may have different dimensions
- Each cell in the grid is called a corner
- Karel can be positioned on corners

Karel's facing directions

- Karel can only be facing one of the four directions



North

South

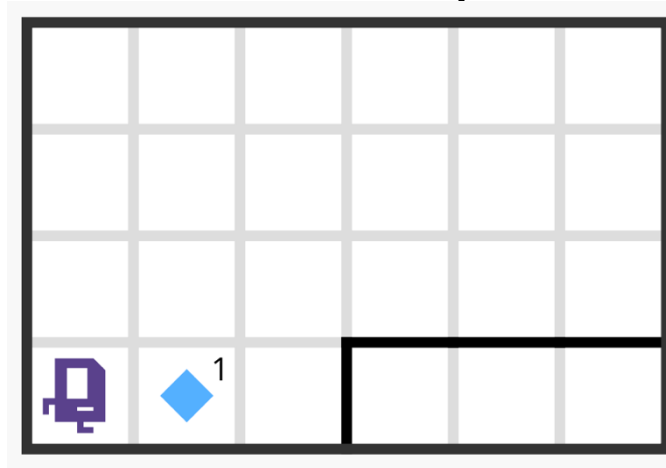
East

West

- Karel's direction is important to determine where it will move next

Beeper and walls

- A corner might have an object called **beeper**
- Karel can only interact with a beeper if they are on the same corner



- The black solid lines are **walls**
- Walls are barriers that Karel cannot walk through

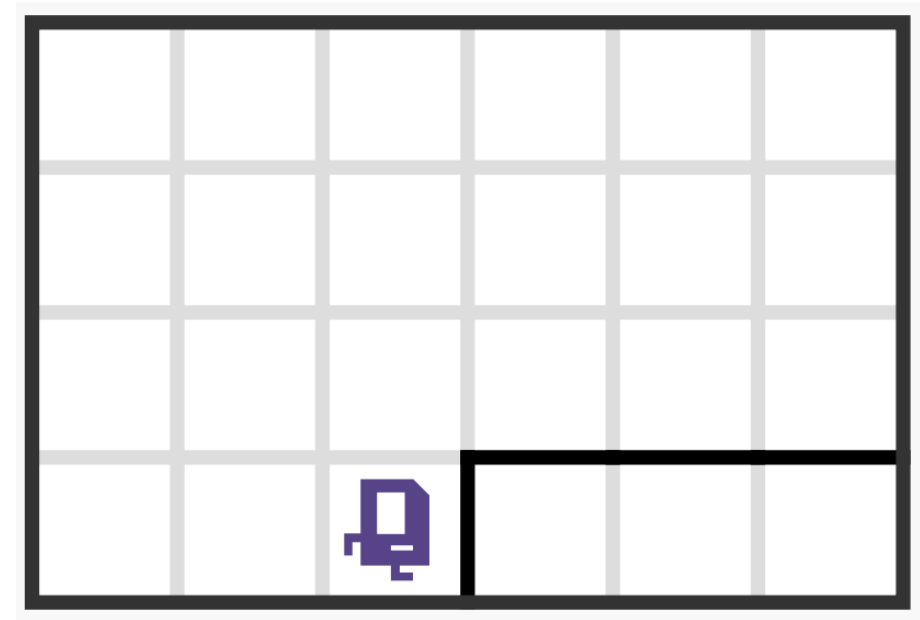
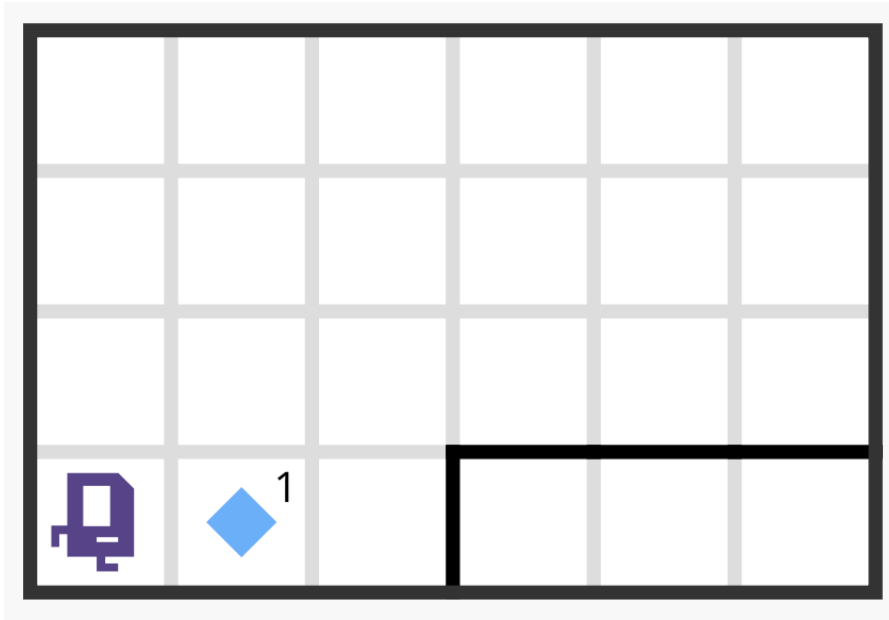
Karel's commands

Command	Effect
<code>move()</code>	Move forward by one corner. If there is a wall blocking its way, an error condition will occur .
<code>turn_left()</code>	Rotate 90 degrees to the left (counterclockwise).
<code>pick_beeper()</code>	Pick a beeper from the corner into its <i>unlimited</i> bag. If there is no beeper in the current corner, an error condition will occur .
<code>put_beeper()</code>	Take a beeper from Karel's bag and put it down in the current corner. If there is no beeper in its bag, an error condition will occur .

Pair of parentheses in each command is a common syntax shared by Karel and Python language. It's used to run/execute a command. Don't forget them!

Karel first program

How can you program Karel to move forward twice and pick up the beeper along the way?

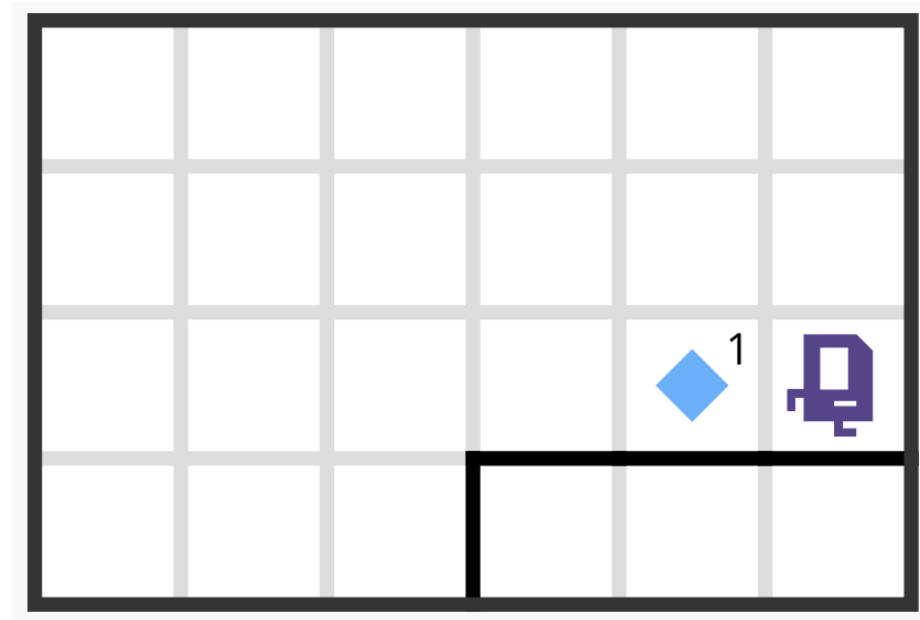
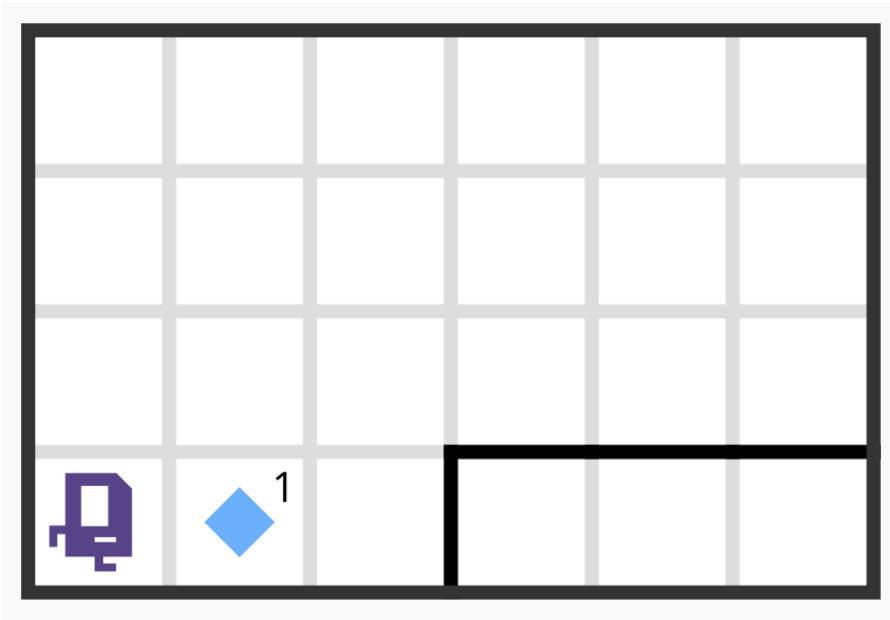


Notes on the Karel program

- Karel programs MUST include the import statement at the beginning and define a function called `main()`
- General Python programs, however, are more flexible and do not have to follow this particular structure
- The `main()` function is the entry point of the program. When running the program, Karel will execute the commands in the `main()` function
- The lines starting with `#` are comments, i.e., text designed to explain the operation of the program to human readers.

Karel second program

How to move the beeper and Karel like this?



Outline

- Meet Karel the robot
- **Libraries, modules, namespaces**
- Functional abstraction & decomposition
- The Python language: first steps

Libraries, modules

- **Library** is a generic term for a collection of (useful) functions, data structures, etc.
- In python, libraries are called **modules**
- One way of **importing** a module is as follows:

```
import math          # math is a built-in module
import standfordkarel
import my_module      # a self-written module
```

which makes the module contents available to use in the program

Modules and namespaces

- Imported names are prefixed with the module name, e.g., **math.pi** provides irrational number π (3.14159...)
 - These names are placed in a separate **namespace**. For example, “pi” is placed in namespace “**math**”.
- How does Python find modules?
 - Standard modules (e.g., math) are installed in a specific location on the file system
 - Non-standard modules (e.g., my_module) must be in the current working directory.
- You can also import all definitions from a module into a program:

```
from math import *
```

Here you can write “pi” directly in the program, without having to write “math.pi”.

Outline

- Meet Karel the robot
- Libraries, modules, namespaces
- **Functional abstraction & decomposition**
- The Python language: first steps

Functional abstraction & decomposition

- In programming, a **function** (also known as “procedure” or subroutine”) is a piece of the program that is given a name
 - The function is called by its name
 - A function is defined once, but can be called many times
- In the Karel programs so far, move, turn_left, pick_beeper, and put_beeper are examples of functions

Why using functions?

- Abstraction: To use a function, we only need to know **what it does, not how**.
- Break a complex program into smaller parts (known as **functional decomposition**).



Defining a function in Python

```
# Turns Karel 90 degrees to the right
def turn_right():
    turn_left()
    turn_left()
    turn_left()
```

- **def** is a Python keyword (“reserved word”)
- **function name** is followed by a pair of parentheses and a colon
 - Inside the parentheses are function’s parameters (more on this later)
- The **function suite** is a sequence of **statements** that will be executed when the function is called.
- All statements in the suite **must be indented by the same number of spaces** (standard is 4 spaces).

Outline

- Meet Karel the robot
- Libraries, modules, namespaces
- Functional abstraction & decomposition
- **The Python language: first steps**

Syntax

- The **syntax** of a programming language is the rules that define what is a valid program
- A python program is a sequence of statements:

- define a function:

```
def turn_around():  
    turn_left()  
    turn_left()
```

- call a function:

```
turn_around()
```

- import a module:

```
import math
```

- more later...

Whitespace

- Spaces, tabs, and end-of-line are known as **whitespace**
- The whitespace before a statement is called **indentation**
- In Python, whitespace has two special roles:
 - end-of-line marks the end of a statement
 - indentation defines the extent of a **suite** of statements
- Other than this, white space is ignored

Permitted names in Python

- A name in Python may contains letters, numbers and numberscores (`_`), but must begin with a letter or underscore.

Allowed	Not allowed
<code>turn_right</code>	<code>turn right</code>
<code>turn_right_2</code>	<code>2_turn_right</code>
<code>is_good</code>	<code>is_good?</code>
<code>imPort</code>	<code>import</code>

- Reserved words cannot be used as names
- Names are **case sensitive**: upper and lower case letters are not the same.

Comments

- A hash sign (#) marks the beginning of a comment; it continues to the end-of-line
- Comments are ignored by Python:
 - Comments are for people to read
 - Use comments to state what is not obvious from the program.
- *If it was hard to write, it's probably hard to read. Add a comment.*