COMP1730/COMP6730
Programming for Scientists

Functional abstraction with Karel the robot

Some announcements

* Two new labs open on Thursday 8-10am
* Doing lab exercises is very important in this course, even more than lectures! **You are strongly encouraged to participate in labs from next week**
* Using AI tools such as ChatGPT and Copilot is OK for everything **except assignments and exam**.
* Recommended text books:
  - Think Python. Allan Downey, *O'Reilly, 2015*
  - A Primer on Scientific Programming with Python, Hans Petter Langtangen, *Springer, 2017*

Lecture outline

* **Meet Karel the robot**
* Libraries, modules, namespaces
* Functional abstraction and decomposition
* The python language: First steps

History behind Karel the robot

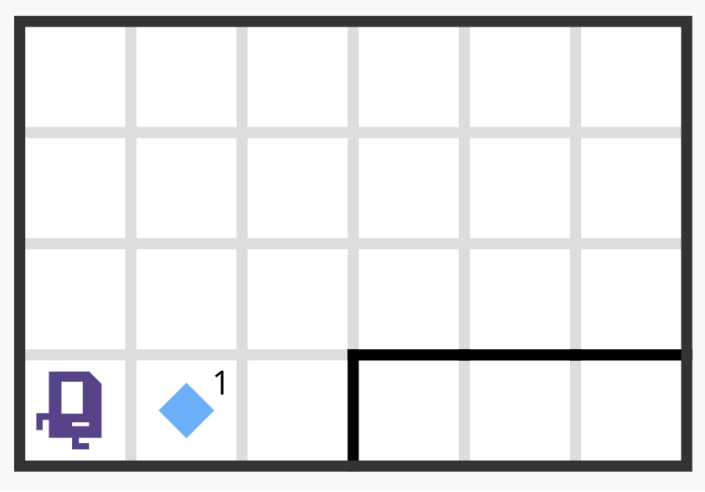
* Gentle **introductory programming environment** proposed by Richard Pattis (graduate student at Stanford Uni) in the 1970s to learn how to problem solve using computers
* Well-received by millions of students worldwide
* “Karel language” is much simpler than Python and other programming languages
* You will “teach” (**program**) 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 very reduced set of predefined commands, but **a key part of programming is defining new commands that extend its initial capabilities**
* Karel language is a much simplified version of Python.

Karel's world (I)

* 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 world (II)

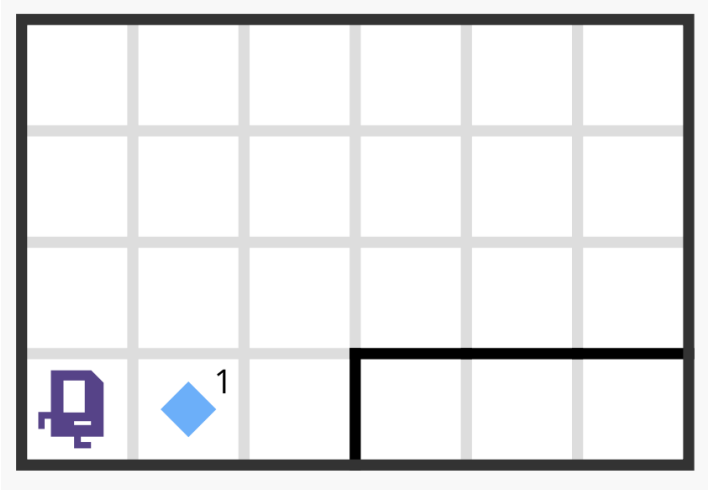
* Karel can only be facing one of the four directions

North South East West

* As shown later, which direction Karel is facing is important because it determines the direction in which Karel will move when commanded to move
* Which direction is Karel is facing in the previous slide?

Karel's world (III)

* A corner might have an object called **beeper**
* Karel can only interact with a beeper if it is on the same corner



* The black solid lines in the diagram are **walls**
* Walls are barriers that Karel cannot walk through walls

Karel's commands (I)

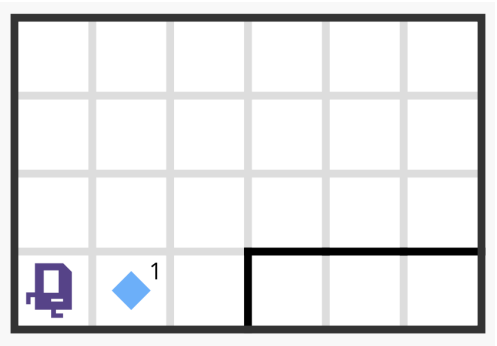
Command	Description
`move()`	Asks Karel to move forward one corner. Karel cannot respond to a `move()` command if there is a wall blocking its way
`turn_left()`	Asks Karel to rotate 90 degrees to the left (counterclockwise)
`pick_beeper()`	Asks Karel to pick up one beeper from a corner and stores the beeper in its beeper bag, which can hold an infinite number of beepers. Karel cannot respond to a `pick_beeper()` command unless there is a beeper on the current corner
`put_beeper()`	Asks Karel to take a beeper from its beeper bag and put it down on the current corner. Karel cannot respond to a `put_beeper()` command unless there are beepers in its beeper bag

Karel's commands (II)

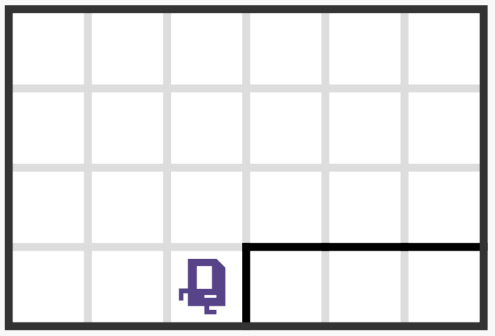
* Empty pair of parentheses in each command is part of the common syntax shared by Karel and Python and is used to specify the invocation of the command (**don't forget them!**)
* If Karel tries to do something illegal, such as moving through a wall or picking up a nonexistent beeper, **an error condition occurs** (more on this later)
* Karel's commands are not executed on their own. We need to incorporate them into a Karel program (more on this later)

Our first Karel program

```
# Include all definitions from the
# stanfordkarel library
from stanfordkarel import *

# Define a "main" function with the
# commands we want Karel to execute
def main():
    # move Karel forward by one corner
    move()

    # pick up a beeper from current corner
    pick_beeper()

    # move Karel forward by one corner
    move()
```



Before



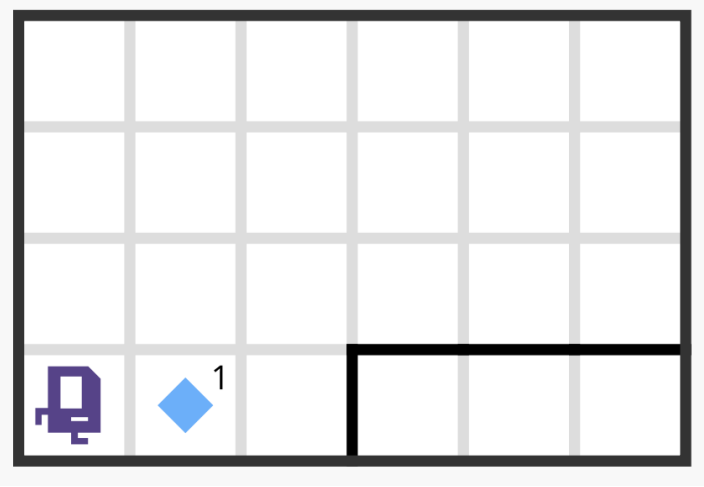
After

Notes on the Karel program

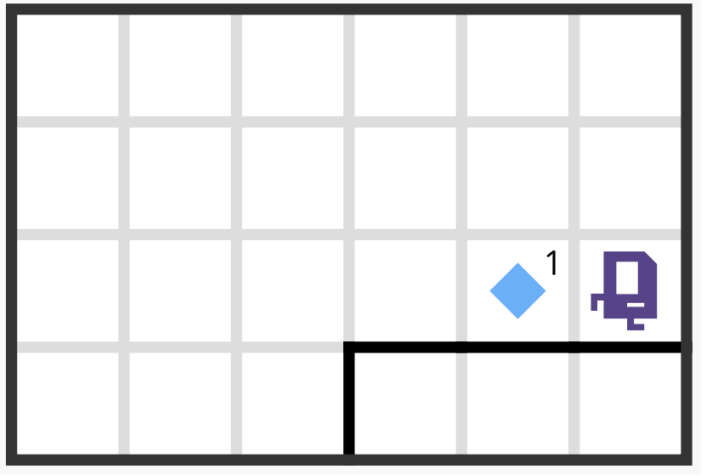
* Karel programs **MUST** include the `import` statement at the beginning and define a function called `main()`
* General Python programs, however, are more flexible and do not have to follow this particular structure
* The `main()` function is the entry point of the program. When the program is executed, Karel will start executing 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

Programming problem

Move the beeper from its initial position on 2nd column and 1st row to the center of a ledge (i.e., corner on 5th column and 2nd row)



before



after

Question: how do we turn Karel right if we can only turn it left with `turn_left()`? (Hint: we can turn it left several times in a row)

Solution

```
# Include all definitions from the
# stanfordkarel library
from stanfordkarel import *

# Define a "main" function with the
# commands we want Karel to execute
def main():
    move()
    pick_beeper()
    move()
    turn_left()
    move()
    turn_left()
    turn_left()
    turn_left()
    move()
    move()
    put_beeper()
    move()
```

Lecture outline

* Meet Karel the robot
* **Libraries, modules, namespaces**
* Functional abstraction and decomposition
* The python language: First steps

Libraries, modules, namespaces

* **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   # my_module is our own self-written module
```

which makes the module contents available to use in the program

* Imported names are prefixed with the module name (e.g., `math.pi` provides irrational number π)
  - These names are placed in a separate **namespace** (more about namespaces later in the course)
* 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* (cwd)
* Alternatively, we can also import all definitions from a module into the program's namespace as we did in the Karel program

```
from math import *
from stanfordkarel import *
```

In this case, we can use the functions directly without prefixing them with the module name

Lecture outline

* Meet Karel the robot
* Libraries, modules, namespaces
* **Functional abstraction and decomposition**
* The python language: First steps

Functional abstraction and 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 any number of times
* In the Karel programs so far, `move`, `turn_left`, `pick_beeper`, and `put_beeper` are examples of functions

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

"*Engineering succeeds and fails because of the black box*"
Kuprenas & Frederick, "101 Things I Learned in Engineering School"

Function definition 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 the function’s parameters (more on this in coming lectures)
* The **function suite** is the 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/tabs** (standard is 4 spaces)

Lecture outline

* Meet Karel the robot
* Libraries, modules, namespaces
* Functional abstraction and 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**:
  - defining a function:
```
def turn_around():
    turn_left()
    turn_left()
```
  - calling a function:
```
put_beeper()
turn_around()
```
  - importing a module:
```
import math
```
  - ...and a few more.

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 (some exceptions, more later in the course)
  - indentation defines the extent of a **suite** of statements
* Other than this, whitespace is ignored

Permitted names in python

* A function name in python may contain letters, numbers and underscores (_), but must begin with a letter or undescore

Allowed	Not allowed
`turn_right`	`turn right`
`turn_right_2`	`2_turn_right`
`is_good`	`is_good?`
`imPort`	`import`

* 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 end-of-line
* Comments are ignored by the interpreter
  - Comments are for **people**
  - Use comments to state what is not obvious
* *If it was hard to write, it's probably hard to read. Add a comment.*