

COMP1730/COMP6730

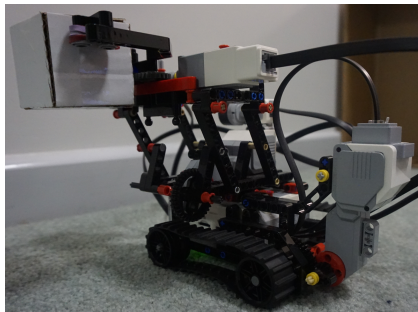
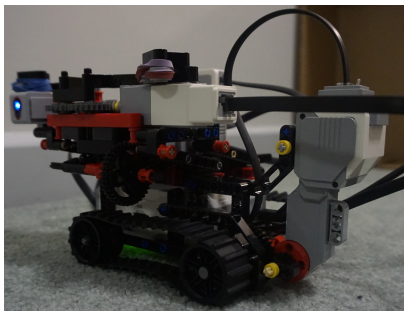
Programming for Scientists

Functional abstraction,
with robots

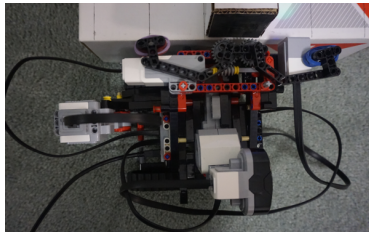
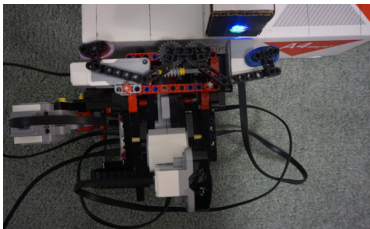
Lecture outline

- * The warehouse robot
- * Functional abstraction
- * The python language: First steps.

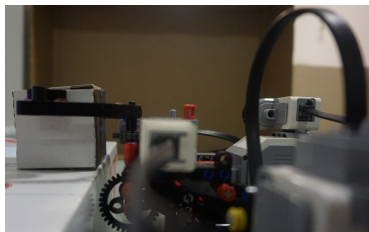
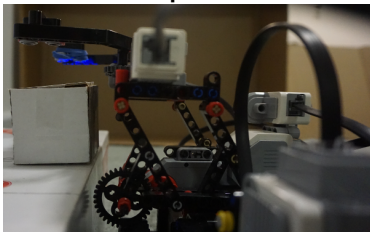
The robot



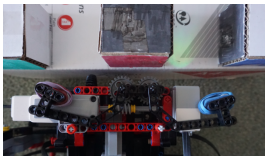
- * Drive left/right along the shelf:



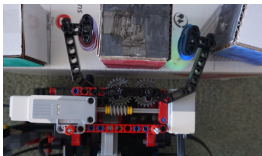
- * Move lift up/down:



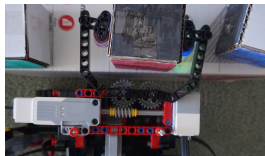
- * Change position of the gripper:



folded



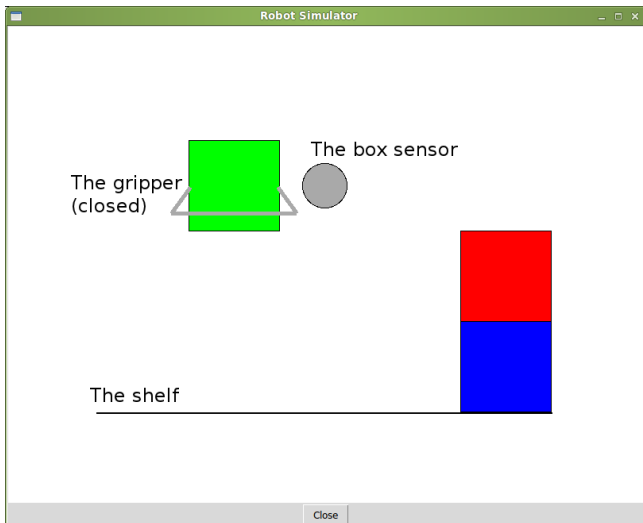
open



closed

- * Moving sideways or down, the gripper may hit boxes if it is not folded.
- * Folding/unfolding the gripper may hit boxes in adjacent stacks.

The robot simulator



```
>>> import robot
```

Start new simulation:

```
>>> robot.init()
```

Start simulation with larger area:

```
>>> robot.init(width = 11, height = 6)
```

Start simulation with random boxes:

```
>>> robot.init(width = 11, height = 6,  
               boxes = "random")
```

Drive right/left one step:

```
>>> robot.drive_right()
```

```
>>> robot.drive_left()
```

Move the lift up one step:

```
>>> robot.lift_up()
```

Move the lift down one step:

```
>>> robot.lift_down()
```

Change gripper position:

```
>>> robot.gripper_to_open()
```

```
>>> robot.gripper_to_closed()
```

```
>>> robot.gripper_to_folded()
```

- * If the robot hits a box, no command works until a new simulation is started.





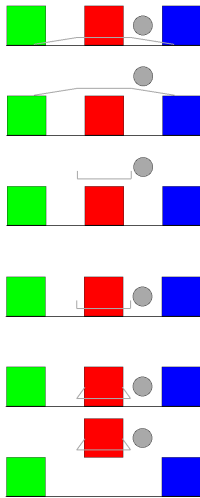
- * How to pick up a box without hitting the box next to it?



- * How to pick up a box without hitting the box next to it?

```
robot.lift_up()  
robot.gripper_to_open()  
robot.lift_down()  
robot.gripper_to_closed()  
robot.lift_up()
```

- * A *program* is a sequence of instructions.



Libraries, modules, namespaces

- * *Library* is a generic term for a collection of (useful) functions, data structures, variables, etc.
- * In python, libraries are called *modules*.
- * *Importing* a module,

```
import math  
import robot
```

makes its content available to use.



- * Imported names are prefixed with the module name, as in `math.pi`, `robot.lift_up`, etc.
 - They 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., `robot`) must be in the *current working directory* (cwd).

```
>>> import os
```

```
>>> os.getcwd()
```

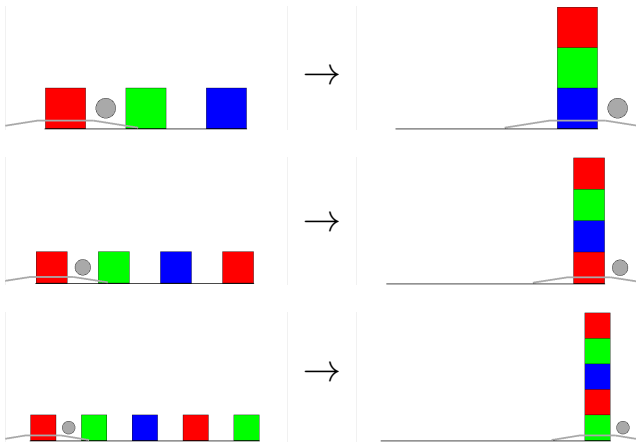
```
'/Users/jeffrey/Desktop/COMP_1730'
```

- * When running a program, the cwd is (normally) the directory where the file is.



Programming and functional abstraction

Problem: Building a tower





```
robot.init(width = 7, boxes = "flat")
robot.drive_right()
robot.lift_up()
robot.gripper_to_open()
robot.lift_down()
robot.gripper_to_closed()
robot.lift_up()
robot.drive_right()
robot.drive_right()
robot.gripper_to_open()
robot.lift_down()
robot.gripper_to_closed()
robot.lift_up()
robot.drive_right()
robot.drive_right()
robot.gripper_to_open()
robot.lift_down()
:
```


Functional abstraction

- * In programming, a *function* (also “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.
- * Why functions?
 - **Abstraction:** To use a function, we only need to know *what* it does, *not how*.
 - Break a complex problem into smaller parts.

Function definition in python

```
def move_to_next_stack():  
    robot.drive_right()  
    robot.drive_right()
```

name
suite

- * `def` is a python keyword (“reserved word”).
- * The *function's 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.

Function definition in python

```
def grasp_box_on_shelf():  
    robot.lift_up()  
    robot.gripper_to_open()  
    robot.lift_down()  
    robot.gripper_to_closed()  
    robot.lift_up()
```

4
spaces

- * In python, a suite is delimited by *indentation*.
 - All statements in the suite **must be preceded by the same number of spaces/tabs** (standard is 4 spaces or 1 tab).

Function definition in python

```
def release_and_pickup_next():  
    robot.gripper_to_open()  
    robot.lift_down()  
    robot.gripper_to_closed()  
    robot.lift_up()
```

- * The `def` statement only *defines* the function – it does not execute the suite.
- * The whole definition is itself a statement.

Building a tower of 5 boxes

```
robot.init(width = 9, boxes = "flat")
robot.drive_right()
grasp_box_on_shelf()
move_to_next_stack()
release_and_pickup_next()
move_to_next_stack()
release_and_pickup_next()
move_to_next_stack()
release_and_pickup_next()
move_to_next_stack()
robot.gripper_to_folded()
robot.lift_down()
```



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 move_twice():  
    robot.drive_right()  
    robot.drive_right()
```
- calling a function:

```
move_twice()  
robot.lift_up()
```
- importing a module: `import robot`
- ...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);
 - 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 underscore.

Allowed

`moverighttwice`

`move_right_2`

`is_box_red`

Not allowed

`move right twice`

`2_steps_right`

`is_box_red?`

`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.

```
robot.init(width = 7) # use a wider shelf
# grasp the first box:
robot.lift_up()
...
```

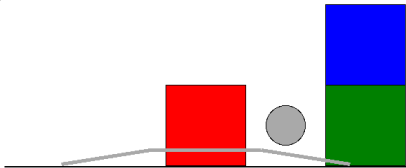
- * Comments are ignored by the interpreter.
 - Comments are for *people*.
 - Use comments to state what is not obvious.
- * If it was hard to write, it's probably hard to read. Add a comment. (Punch & Enbody, Rule 6)

- * Write comments to describe *what* a function does, and *when* it should be expected to work.

```
# Pick up a box from the shelf, without
# hitting adjacent boxes.
# Assumptions: The robot (gripper) is in
# front of the box; the gripper is folded
# and the lift is down.
def grasp_box_on_shelf():
    ...
```

Test, test, test

- * How do we know our program works?
 - Specify the assumptions under which the program (or function) is meant to work.
 - Test it with a variety of cases that fall under those assumptions.
 - Particularly, “edge cases”.



Errors

```
Traceback (most recent call last):  
  File "stack-3-v1.py", line 35, in <module>  
    robot.lift_up()  
  File "/.../robot.py", line 40, in lift_up  
    _robot.lift_up()  
  File "/.../robot.py", line 600, in lift_up  
    + " and can't go any higher!")  
robot.RobotError: Robot Error: The lift is at  
level 1 and can't go any higher!
```

- * Errors will happen.
- * Read the error message!

* Some common errors:

- `SyntaxError`:

You have broken the rules of python syntax.

- `NameError` or `AttributeError`:

You have used a (function) name that doesn't exist. Check for typos.

- `IndentationError`:

Too much or too little indentation.

- All statements in a suite must have the same indentation.

- All statements in main program must have no indentation.