Australian National University

COMP1730/COMP6730
Programming for Scientists

Control, part 1: Branching

Homework

* Homework 1
  - Due at 9am (Canberra Time), Monday 15 March.
  - Marking in *your* lab next week (Week 4).
  - Please carefully read the submission instructions.
* Homework 2
  - Deadline is **9:00am Monday the 22nd March**.

Course Contact Details

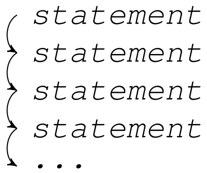
* Wattle forums for questions on the course content.
* E-mail to comp1730@anu.edu.au for personal matters.
* Ask your tutor in lab groups.
* You can find the code to sign into a Teams group in Wattle.
* Catch-up labs Fridays 11:00am-1:00pm and 1:00pm-3:00pm online and in HN1.23.

Outline

* Program control flow
* Branching: The `if` statement
* Examples

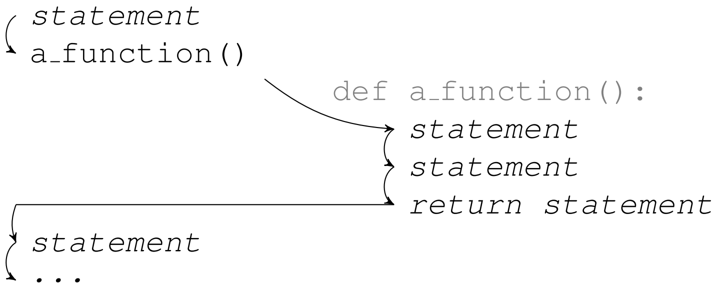
Program control flow

Sequential program execution



```
statement
statement
statement
statement
...
```

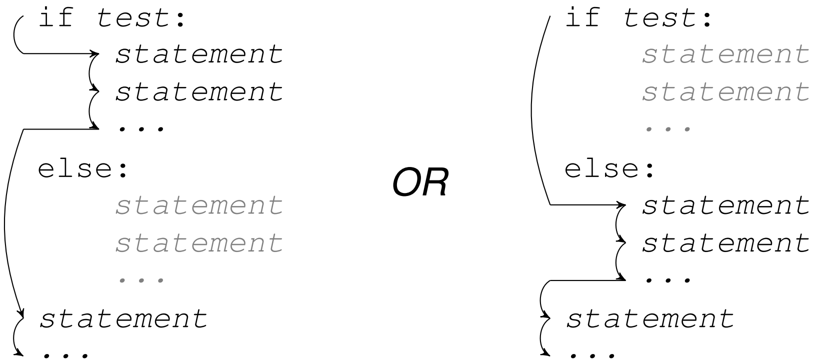
- The python interpreter always executes instructions (statements) one at a time in sequence.



```
statement
a_function()
                    def a_function():
                        statement
                        statement
                        return statement
statement
...
```

* Function calls “insert” a function suite into this sequence, but the sequence of instructions remains invariably the same.

Branching program flow



```
if test:
    statement
    statement
    ...
else:
    statement
    statement
    ...
statement
...
```

OR

```
if test:
    statement
    statement
    ...
else:
    statement
    statement
    ...
statement
...
```

* Depending on the outcome of a test, the program executes one of two alternative branches.

The `if` statement

```
if test_expression :
    suite
statement(s)
```

1. Evaluate the test expression (converting the value to type `bool` if necessary).
2. If the value is `True`, execute the suite, then continue with the following statements (if any).
2. If the value is `False`, skip the suite and go straight to the following statements (if any).

The `if` statement, with `else`

```
if test_expression :
    suite_1
else:
    suite_2
statement(s)
```

1. Evaluate the test expression.
2. If the value is `True`, execute suite #1, then following statements (if any).
2. If the value is `False`, execute suite #2, then following statements (if any).

Truth values (reminder)

- Type `bool` has two values: `False` and `True`.
- Boolean values are returned by comparison operators (==, !=, <, >, <=, >=) and a few more.
- Ordering comparisons can be applied to pairs of values of the same type, for (almost) any type.
- *Warning #1*: Where a truth value is required, python automatically converts any value to type `bool`, but it may not be what you expected.
- *Warning #2*: Don't use arithmetic operators (+, −, ∗, etc.) on truth values.

Suites (reminder)

- A *suite* is a (sub-)sequence of statements.
- A suite must contain at least one statement!
- 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).
  - The indentation depth of the suite inside an `if` (and `else`) statement must be greater than that of the `if` (`else`).
- A suite can include nested suites (`if`'s, etc).

Suites: A side remark

- (Almost) Every programming language has a way of grouping statements into suites/blocks.
  - For example, in C, Java and many other:

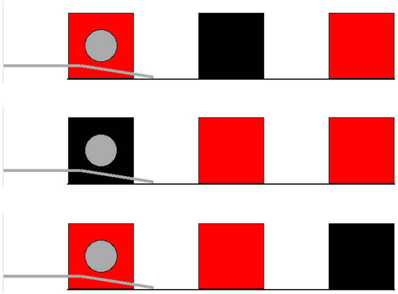
    ```
    if (expression) {
      suite
    }
    ```
  - or in Ada or Fortran (post -77):

    ```
    if expression then
      suite
    end if
    ```
- The use of indentation to *define* suites is a python peculiarity.

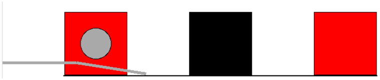
Examples

Problem: Stack the red boxes

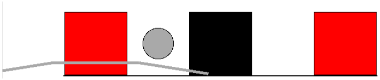
- Two of three boxes on the shelf are red, and one is not; stack the two red boxes together.
- Write a program that works wherever the red boxes are.



- `robot.sense_color()` returns the color of the box in front of the sensor, or no color (`''`) if no box detected.



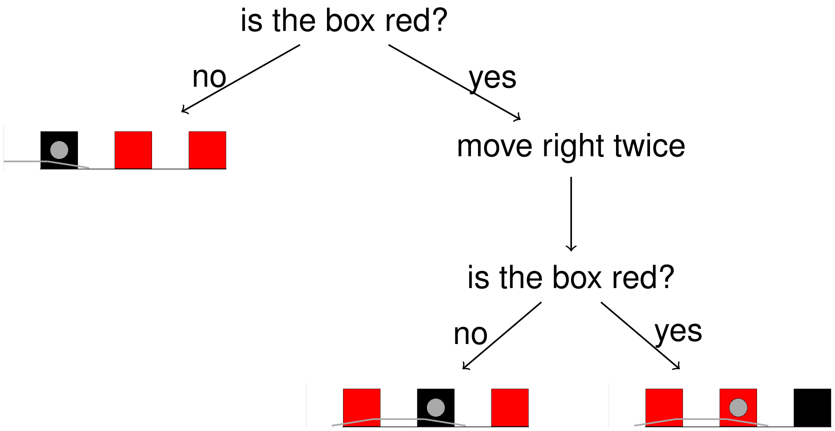
```
>>> robot.sense_color()
'red'
```



```
>>> robot.sense_color()
''
```

  - Note that the color name is a string (in `''`)
  - The box sensor is one step right of the gripper (it’s the circle in the simulator).

Algorithm idea



```
def stack_red_boxes():
    if robot.sense_color() == 'red':
        drive_right_twice()
        if robot.sense_color() == 'red':
            # stack middle box on left
        else:
            # stack left box on right
    else:
        # stack middle box on right
```

```
def print_grade(mark):
    if mark >= 80:
        print('HD')
    if mark >= 70:
        print('D')
    if mark >= 60:
        print('Cr')
    if mark >= 50:
        print('P')
    if mark < 50:
        print('Fail')
```

* What will `print_grade(90)` print?

Boolean operators

- The operators `and`, `or`, and `not` combine truth values:

`a and b`	`True` iff *a* and *b* both evaluate to `True`.
`a or b`	`True` iff at least one of *a* and *b* evaluates to `True`.
`not a`	`True` iff *a* evaluates to `False`.

- Boolean operators have lower precedence than comparison operators (which have lower precedence than arithmetic operators).

```
def print_grade(mark):
    if mark >= 80:
        print('HD')
    if mark < 80 and mark >= 70:
        print('D')
    if mark < 70 and mark >= 60:
        print('Cr')
    if mark < 60 and mark >= 50:
        print('P')
    if mark < 50:
        print('Fail')
```

The `if–elif–else` statement

```
if bool_exp_1 :
    suite_1
elif bool_exp_2 :
    suite_2
elif bool_exp_3 :
    suite_3
 .
 .
else:
    else_suite
statement(s)
```

- Tests are evaluated in sequence, and only the suite corresponding to the first test that returns `True` is executed.
- The `else` suite is executed only if all tests return `False`.

```
def print_grade(mark):
    if mark >= 80:
        print("HD")
    elif mark >= 70:
        print("D")
    elif mark >= 60:
        print("Cr")
    elif mark >= 50:
        print("P")
    else:
        print("Fail")
```