

# COMP1730/COMP6730

## Programming for Scientists

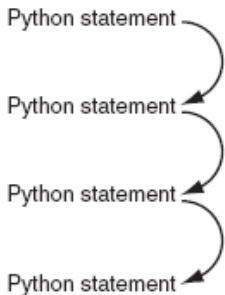
### Control, part 2: Iteration



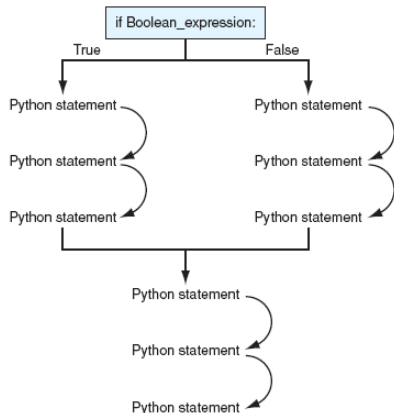
# Outline

- \* Iteration: The `while` statement
- \* Example Problems

# Program control flow



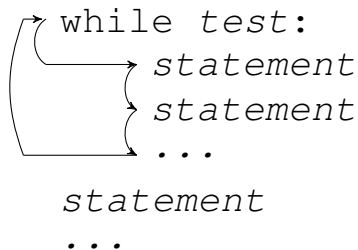
**FIGURE 2.1** Sequential program flow.



**FIGURE 2.2** Decision making flow of control.

Images from Punch & Endbody

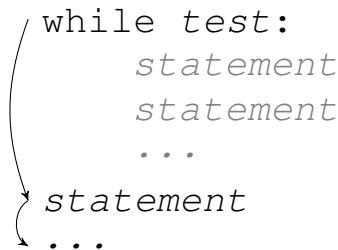
# Iteration (Repetition)



while *test*:  
    *statement*  
    *statement*  
    ...  
*statement*  
...

The diagram shows a code block for a while loop. A curved arrow on the left side starts from the bottom of the loop body and points back to the beginning of the loop, indicating repetition.

*UNTIL*



while *test*:  
    *statement*  
    *statement*  
    ...  
*statement*  
...

The diagram shows a code block for an until loop. A curved arrow on the left side starts from the bottom of the loop body and points back to the beginning of the loop, indicating repetition.

- \* Iteration *repeats* a suite of statements.
- \* A test is evaluated before each iteration, and the suite executed (again) if it is true.

# Iteration statements in python

- \* The `while` loop repeats a suite of statements as long as a condition is true.
- \* The `for` loop iterates through the elements of a collection or sequence (data structure) and executes a suite once for each element.
  - We'll come back to the `for` loop later in the course.

# The `while` loop statement

```
while 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 once, then go back to **1**.
3. If the value is `False`, skip the suite and go on to the following statements (if any).

# 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 following `if / else / while` : must be greater than that of the statement.
- \* A suite can include nested suites (`if`'s, etc).

# Variable assignment (reminder)

- \* A variable is a name that is associated with a value in the program.
- \* Variable assignment is a statement:  
*var\_name = expression*
  - Note: Equality is written `==` (two `=`'s).
- \* A name–value association is created by the *first* assignment to the name;
- \* *subsequent* assignments to the same name *change* the associated value.



```
→ 1 an_int = 3 + 2  
→ 2 an_int = an_int * 5
```

```
1 an_int = 3 + 2  
→ 2 an_int = an_int * 5
```

Global frame

an\_int | 5

Global frame

an\_int | 25

★ For example,

```
an_int = 3 + 2
```

```
an_int = an_int * 5
```

(From [pythontutor.com](http://pythontutor.com))

1. Evaluate expression `3 + 2` to 5.
2. Store value 5 with name `an_int`
3. Evaluate expression `an_int * 5` to 25.
4. Store value 25 with name `an_int`, replacing the previous associated value.



# Example Problems

# Example: Print numbers

- \* Print out the numbers 1 through 10.

```
n = 1
while n <= 10:
    print(n)
    n = n + 1
```

- \* This is a common `while` loop setup.



## Example: Sums

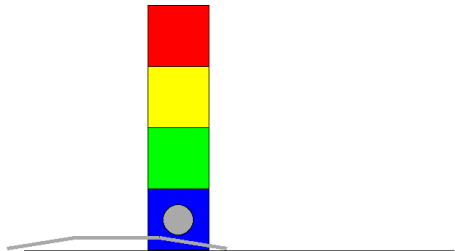
- \* For a given  $n$ , what is the sum of the integers 1 through  $n$ ?

```
def sum_integers(n)
    k = 1
    sum = 0
    while k <= n:
        k = k + 1
        sum = sum + k
    return sum
```

- \* Is this correct? (Test, test, test!)

# Problem: Counting boxes

- \* How many boxes are in the stack from the box in front of the sensor and up?



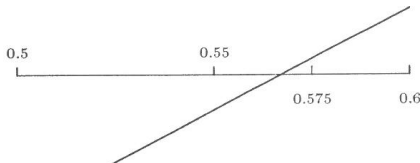
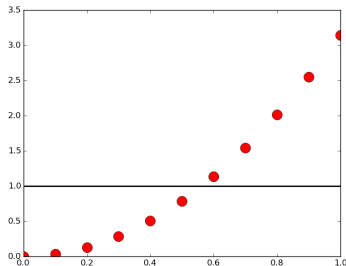
- \* While `robot.sense_color() != ''`, move the lift up, *and count how many times*; then move the lift down that many times.



```
def count_boxes():
    num_boxes = 0
    while robot.sense_color() != '':
        num_boxes = num_boxes + 1
        robot.lift_up()
    steps_to_go = num_boxes
    while steps_to_go > 0:
        robot.lift_down()
        steps_to_go = steps_to_go - 1
    return num_boxes
```

# Problem: Solving an equation

- \* Solve  $f(x) = 0$ .
- \* The interval-halving algorithm:
  - if  $f(m) \approx 0$ , return  $m$ ;
  - if  $f(m) < 0$ , set  $l$  to  $m$ ;
  - if  $f(m) > 0$ , set  $u$  to  $m$ .



# return from a loop

- \* A loop (`while` or `for`) can appear in a function suite, and a `return` statement can appear in the suite of the loop.

```
def find_box(color):  
    while robot.sense_color() != '':  
        if robot.sense_color() == color:  
            return True  
        robot.lift_up()  
    return False
```

- \* Executing the `return` statement ends the function call, and therefore also exits the loop.



# Common problems with `while` loops

- \* Loop never starts: the control variable is not initialised correctly.
- \* Loop never stops (infinite loop): the control variable is not modified in the loop.
- \* Loop runs one too many or one too few times (off by one error).