COMP1730/COMP6730
Programming for Scientists

Control, part 2: Iteration

Outline

* Iteration: The `while` statement with examples
* Common problems with loops.

Program control flow

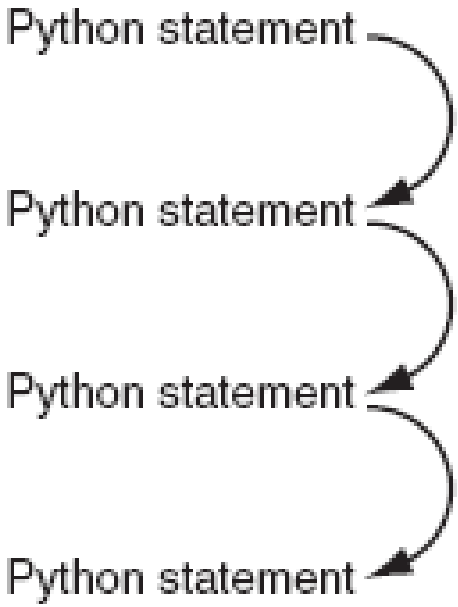


FIGURE 2.1 Sequential program flow.

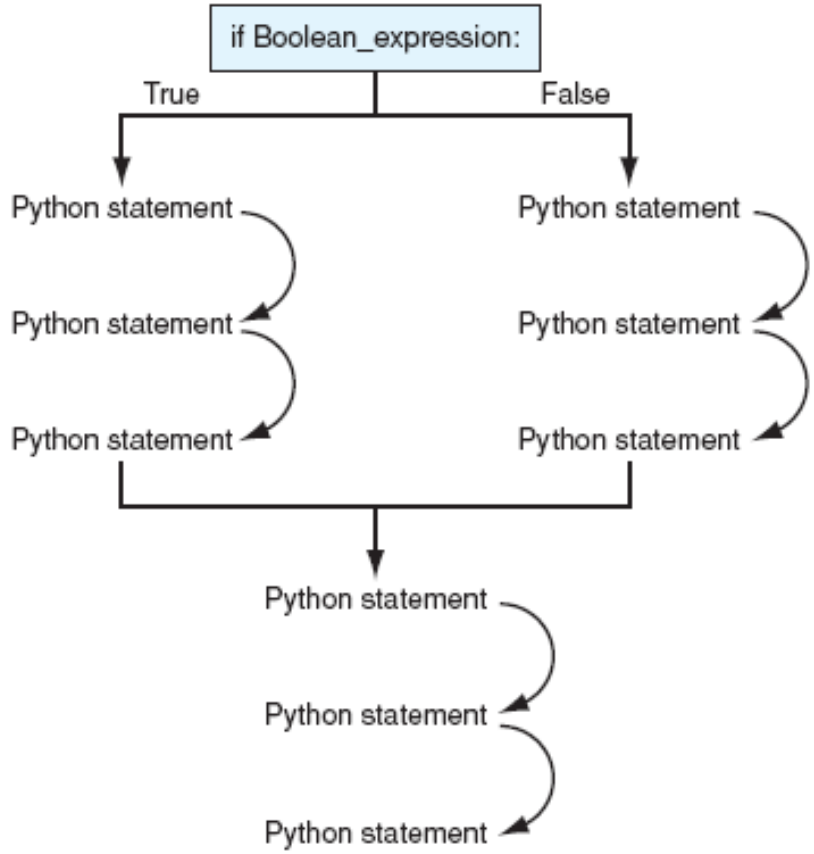
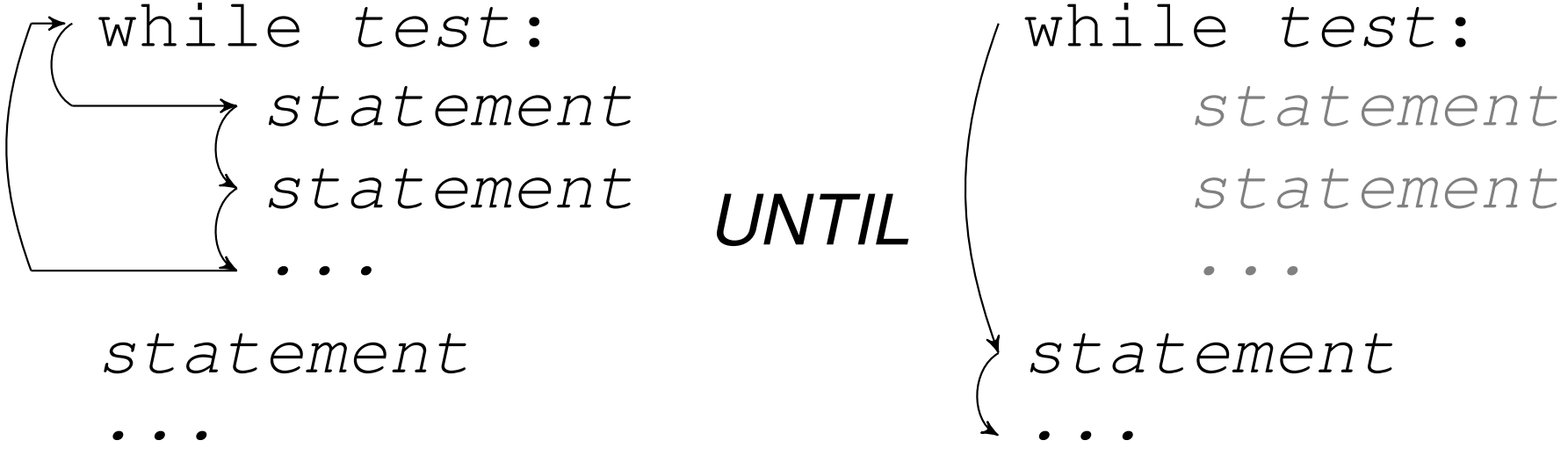


FIGURE 2.2 Decision making flow of control.

Images from Punch & Enbody

Iteration



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

Iteration statements in python

* The `while` loop repeats a block 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 block once for each element. So for loops are most useful for looping a defined number of times, whereas a while statement is best for looping an undefined number of times.
  - See for_loop_examples.py for details on the use of `for` loops.

The `while` loop statement

```
while test_expression :
    block
statement(s)
```

1. Evaluate test expression (converting the value to type `bool` if necessary).
2. If the value is `True`, execute the block once, then go back to **1**.
3. If the value is `False`, skip the block and go on to the following statements (if any).

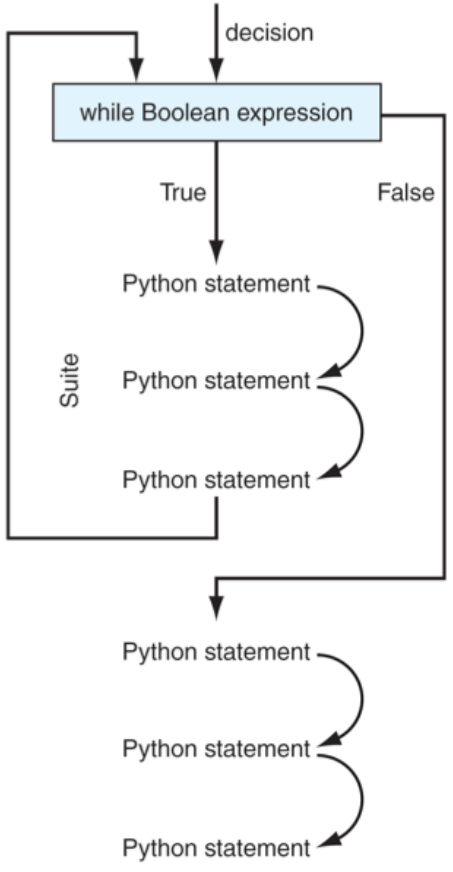


Image from Punch & Enbody

blocks (reminder)

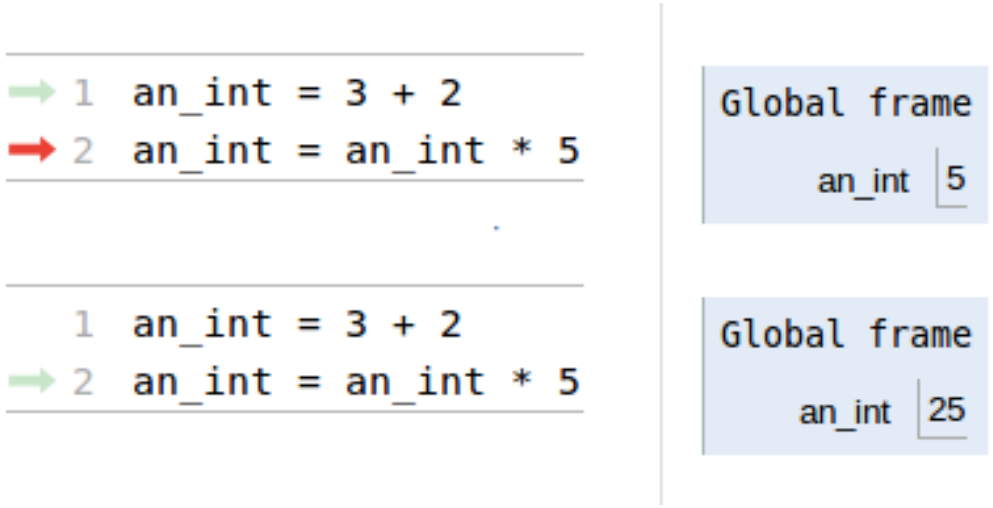
* A *block* is a (sub-)sequence of statements.
* A block must contain at least one statement!
* In python, a block is delimited by indentation.
  - All statements in the block **must be preceded by the same number of spaces/tabs** (standard is 4 spaces).
  - The indentation depth of the block following `if` / `else` / `while` `:` must be greater than that of the statement.
* A block can include nested blocks (`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.



(From `pythontutor.com`)

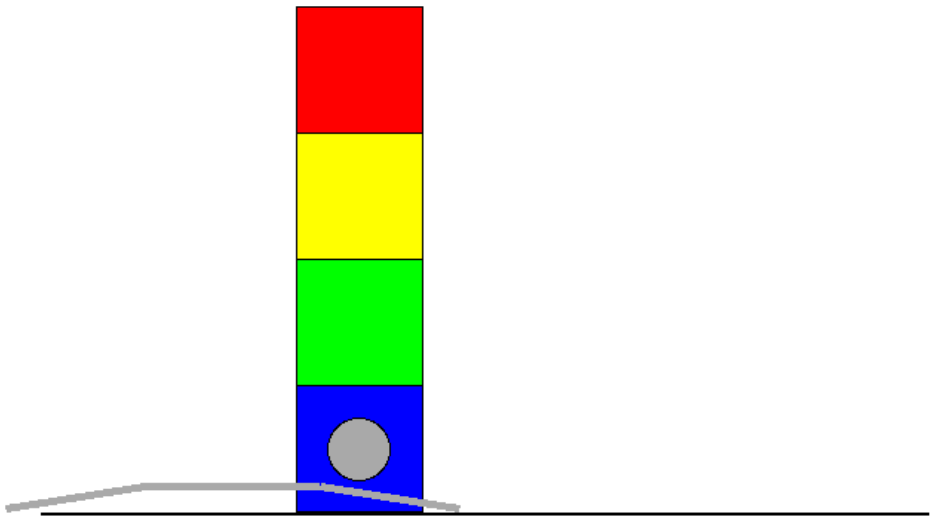
* For example,

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

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.

Problem: Counting boxes

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

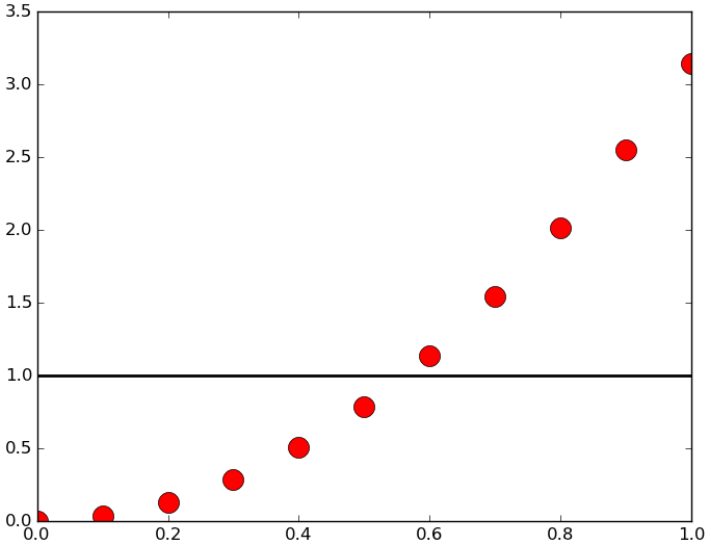


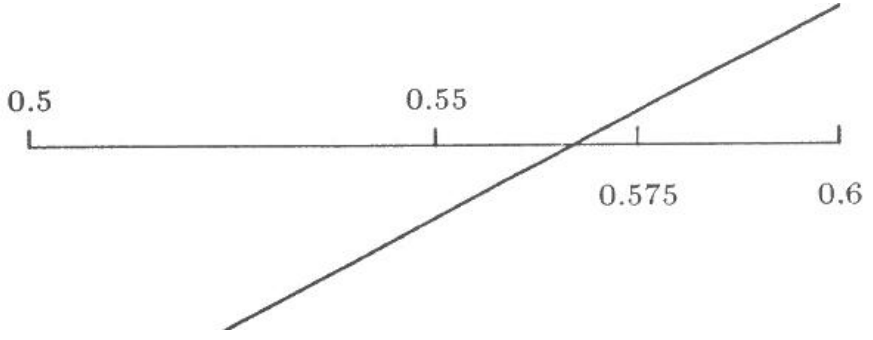
* While `robot.sense_color() != ''`, move the lift up, *and count how many times*; then move the lift down that many times.

```
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 block, and a `return` statement can appear in the block 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.
* (also read up on the `break` and `continue` statements)

Problem: Greatest common divisor

* For two positive integers a and b, find the largest integer that divides a and b.
* Euclid's algorithm: Assuming $a \geq b$,
  - $gcd(a, b) = b$ if b divides a;
  - $gcd(a, b) = gcd(b, a\%b)$, otherwise.

Bounded loops using `for`

* The iteration examples above loop an indefinite number of times (e.g. until convergence).
* It is often the case that we want to loop a fixed number of times.
* This can be done using `while` and a "loop variable"

```
j = 0 # initialise loop variable to 0
while j < 10:
    print(j)
    j = j + 1
```

Bounded loops using `for`

* But a `for` statement is designed for bounded loops and is shorter as it implicitly implements the loop variable

```
for i in range(0,10):
    print(i)
```

* A `for` statement is also widely used to iterate over the contents of a list or other sequence data type (covered next lecture).

```
a = ["one","two","three"] # list of strings
for c in a:
    print(c)
```

Writing and debugging loops

Repeat while condition is true

* A `while` loop repeats as long as the condition (test expression) evaluates to `True`.
* If the condition is initially `False`, the loop executes zero times.
* If no variable involved in the condition is changed during execution of the block, the value of the condition will not change, and the loop will continue forever.

Common problems with `while` loops

* Loop never starts: the control variable is not initialised correctly.

```
# find smallest non-trivial
# divisor of num:
i = 1
while num % i != 0:
    i = i + 1
```

  - `num % 1` is always `0`!

Common problems with `while` loops

* Loop never ends: the control variable is not updated in the loop block, or not updated in a way that can make the condition false.

```
i = 0
while i != stop_num:
    i = i + step_size
```

  - What if `stop_num < 0`?
  - or `step_size < 0`?
  - or `step_size` does not divide `stop_num`?

Take home message

* Branching (`if`) and iteration (`while` loop) are two main control mechanisms to change the sequential flow of a program.
* Some (but not all) recursions can be re-written as iterations to solve the same problem (and vice versa).
* Make sure that the test condition will evaluate to `False` at some point. Otherwise you will enter an infinite loop!