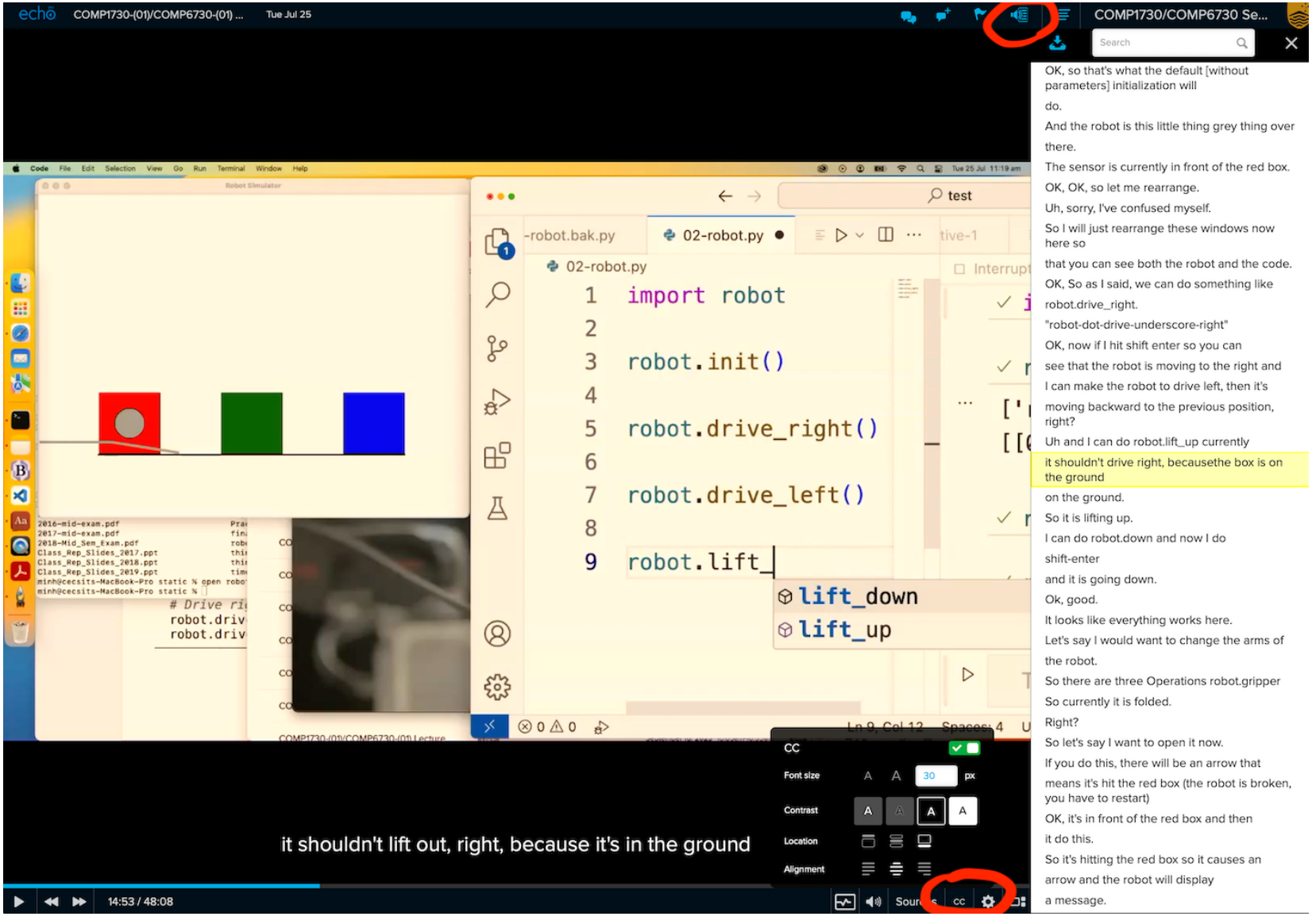
COMP1730/COMP6730
Programming for Scientists

Control, part 2: Iteration

echo360 transcripts and close caption corrected, thanks @Alexei!



Outline

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

Program control flow

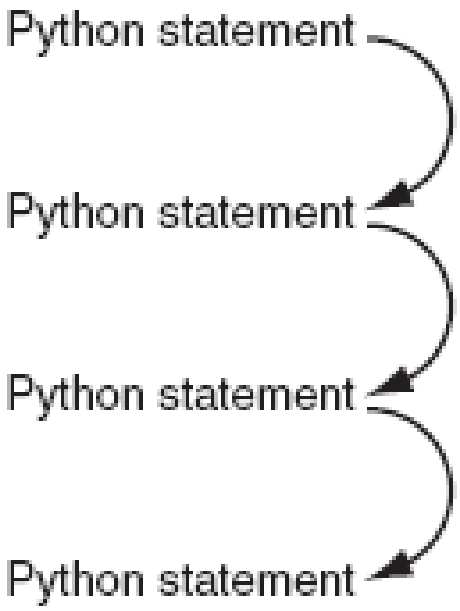


FIGURE 2.1 Sequential program flow.

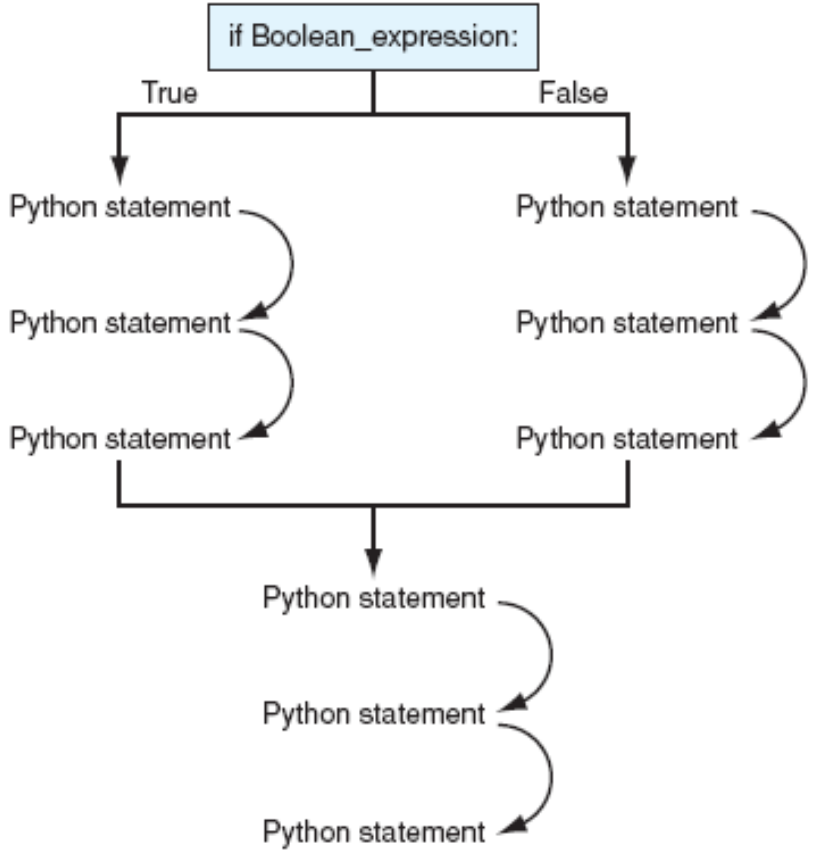
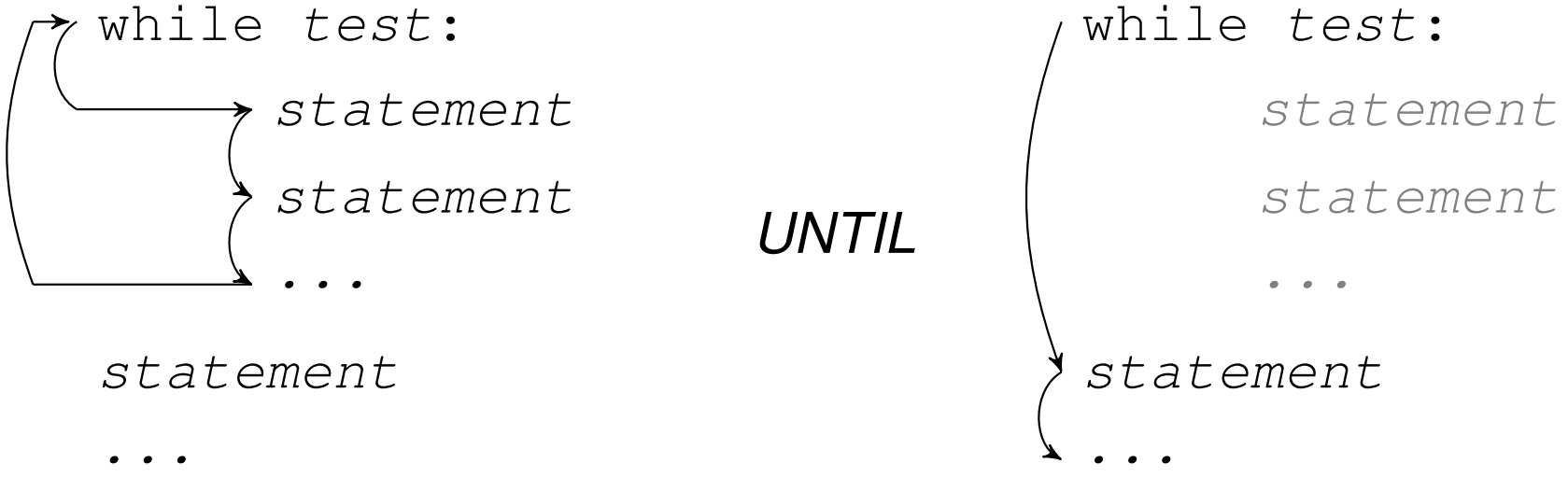


FIGURE 2.2 Decision making flow of control.

Images from Punch & Enbody

Iteration



* Iteration *repeats* a 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

other_statement(s)
```

1. Evaluate 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).

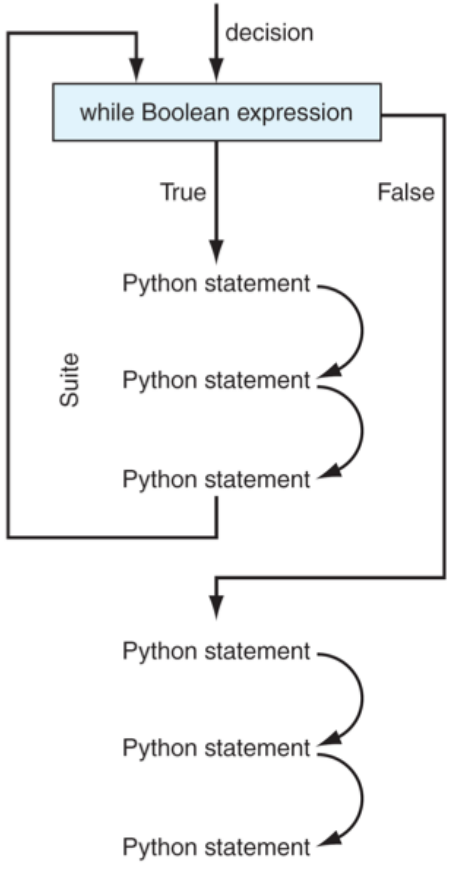


Image from Punch & Enbody

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).
  - 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
```

Global frame

an_int | 5

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

Global frame

an_int | 25

(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.

Example: Sums

* What is the max k such that $(1 + 2 + \ldots + k) \leq 20$?

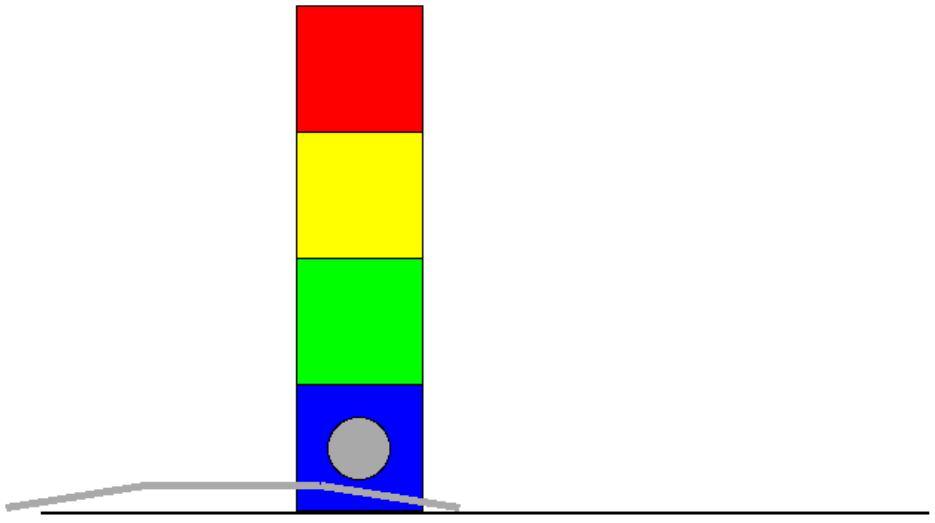
```
k = 1
sum_to_k = 1
while sum_to_k <= 20:
    print("k =", k, ", sum =", sum_to_k)
    k = k + 1
    sum_to_k = sum_to_k + k

print("The answer is", k - 1)
```

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

Coding problem: Counting boxes

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

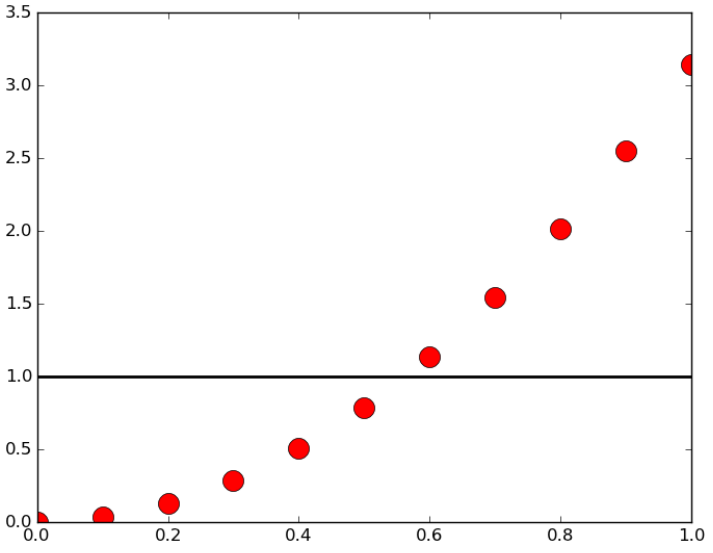


* While `robot.sense_color() != ''`, move the lift up, *and count how many times*.

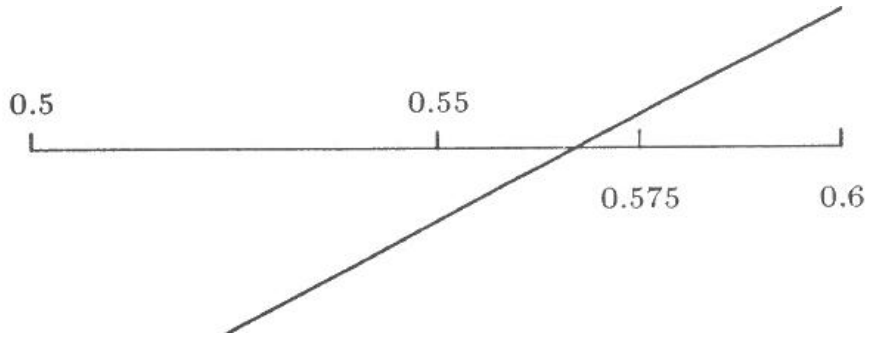
```
def count_boxes_loop():
    num = 0
    while robot.sense_color() != "":
        num = num + 1
        robot.lift_up()
    return num
```

Coding problem: Solving an equation

* Solve $f(x) = 0$.
* For example, find x such that $x^2\pi - 1 == 0$.

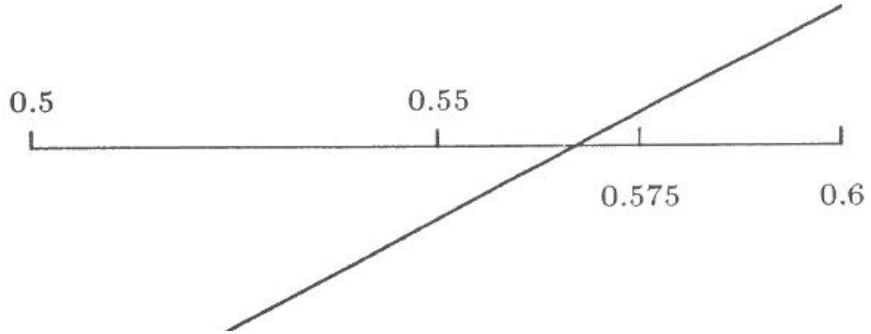


* The interval-halving algorithm.



* Assumption: $f(x)$ is monotone increasing and crosses 0 in the interval $[lower, upper]$.
* Idea:
  - Find the middle of the interval, $m = (lower + upper)/2$:
  - if $f(m) \approx 0$, return m;
  - if $f(m) < 0$, the solution lies between m and $upper$;
  - if $f(m) > 0$, the solution lies between $lower$ and m.

* *Don't compare* `float`*s with* $==$.



Solution using recursion

```
import math

def f(x):
    return x**2 * math.pi - 1

def interval_halving_recursion(lower, upper):
    middle = (lower+upper)/2
    value = f(middle)
    if abs(value) < 1e-6:
        return middle
    elif value < 0:
        return interval_halving_recursion(middle, upper)
    else:
        return interval_halving_recursion(lower, middle)
```

Solution using iteration

```
def interval_halving(lower, upper):
    while upper - lower > 1e-10:
        middle = (lower+upper)/2
        value = f(middle)
        if abs(value) < 1e-6:
            return middle
        elif value < 0:
            lower = middle
        else:
            upper = middle
    return "No value found"
```

`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.

* Question 1: Can all codes using iteration (loop) be rewritten as recursion (function calling itself)?
* Question 2: Can all codes with recursion (function calling itself) be rewritten as iteration (loop)?

PollEverywhere vote results (added after lecture)

Question 1: 81 total votes (correct answer in bold)

* **Yes**: 52 votes
* No: 24 votes (seemed that some people changed their answer after the solution was announced ☺)
* I don't know: 4
* I don't care: 1

Question 2: 76 votes

* **Yes**: 33 votes
* No: 41 votes
* I don't know: 1
* I don't care: 1

M

Can all codes using iteration (loop) be rewritten as recursion (function calling itself)?

Yes, in theory, any code that uses iteration (loops) can be rewritten using recursion (function calling itself), and vice versa. Both iteration and recursion are mechanisms for controlling program flow and achieving repetitive tasks, so it's possible to convert between the two.

However, there are some important considerations to keep in mind:

1. **Stack Depth**: Recursion relies on the call stack, and too much recursion can lead to a stack overflow error. Iteration doesn't have this limitation, making it more suitable for tasks that require deep levels of repetition.
2. **Readability and Maintainability**: While some problems are naturally suited to recursion, others may become less readable when rewritten recursively. Iteration is often more straightforward and easier to understand for many programmers.
3. **Performance**: Recursive solutions can sometimes be less efficient compared to iterative

Source: ChatGPT

M

Can all codes with recursion (function calling itself) be rewritten as iteration (loop)?

Yes, in theory, any code that uses recursion (function calling itself) can be rewritten using iteration (loops). Recursion and iteration are two different ways of achieving repetitive tasks, and they are conceptually equivalent. Any problem that can be solved recursively can also be solved iteratively, and vice versa.

However, similar to the previous answer, there are considerations to take into account:

1. **Stack Overflow**: Recursive solutions can lead to a stack overflow error if not properly managed, especially for deep recursion. Iterative solutions do not have this limitation and can handle a larger number of repetitions.
2. **Complexity**: Some problems are naturally suited to recursion and might become more

Source: ChatGPT

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 suite, the value of the condition will not change, and the loop will continue forever.

Common problems with `while` loops

* What is the problem with the following code?

```
# find smallest non-trivial
# divisor of num
i = 1
while num % i != 0:
    i = i + 1
print("smallest non-trivial divisor of", num,"is", i)
```

* And this?

```
i = 0
while i != 100:
    i = i + 3
```

Common problems with `while` loops

* What is the problem with the following code?

```
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.
* Make sure that the test condition will evaluate to `False` at some point. Otherwise you will enter an infinite loop!