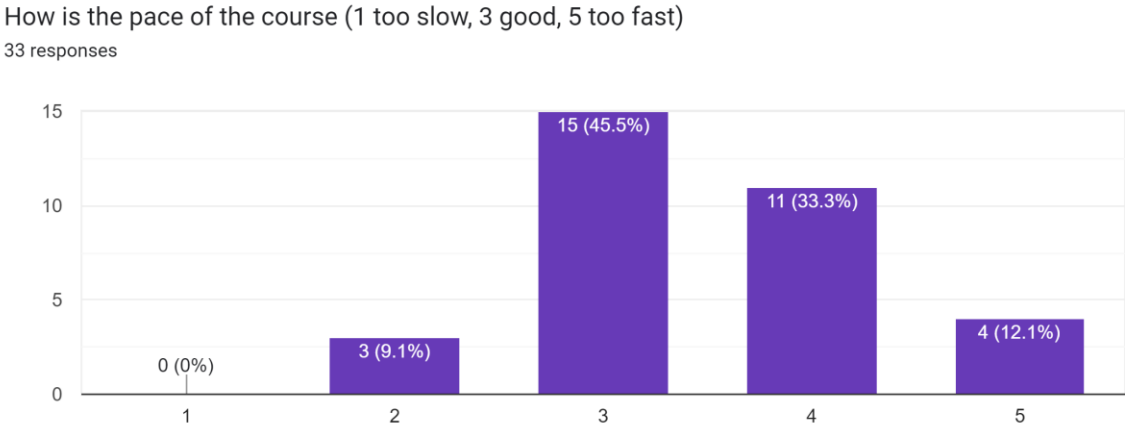
You can access these slides on the course Github:
https://github.com/natrask/ENM1050

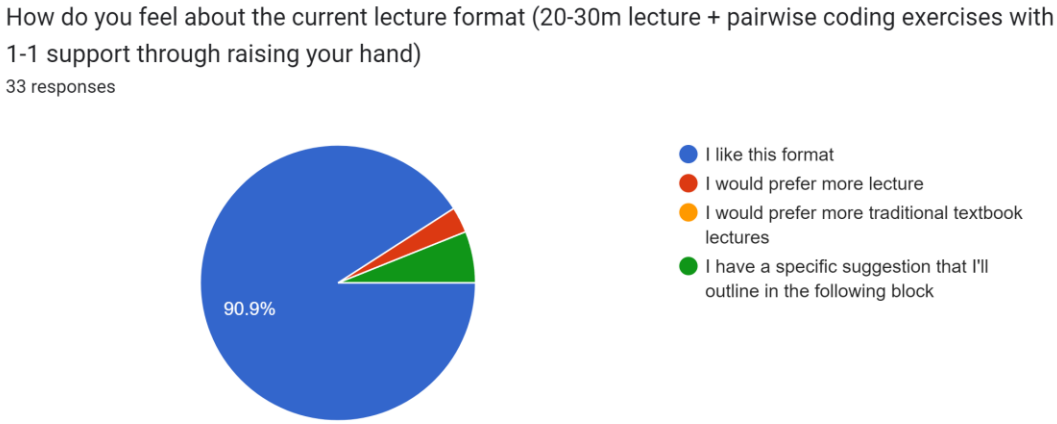
# ENGR 1050
# Intro to Scientific Computation

**Lecture 04 – For and while loops**

Prof. Nat Trask
Mechanical Engineering & Applied Mechanics
University of Pennsylvania

# Course feedback: Pacing, pair coding




# Shorter inclasses, shifting toward scientific apps

# Review from last week

# What is the difference between:

```
val = # some number

if val < 0 :
    a = 0
elif val > 1 :
    a = 1
else :
    a = val
```

```
val = # some number
a = val

if val < 0 :
    a = 0
elif val > 1 :
    a = 1
```

# What is the difference between:

```
val = # some number

if val < 0 :
    a = 0
elif val > 1 :
    a = 1
else :
    a = val
```

For this one, you need to make sure you wrote all the possible paths

```
val = # some number
a = val

if val < 0 :
    a = 0
elif val > 1 :
    a = 1
```

This one is safer

# Last Week:

```
if CONDITION1:
    BODY1
elif CONDITION2:
    BODY2
else:
    BODY3
```

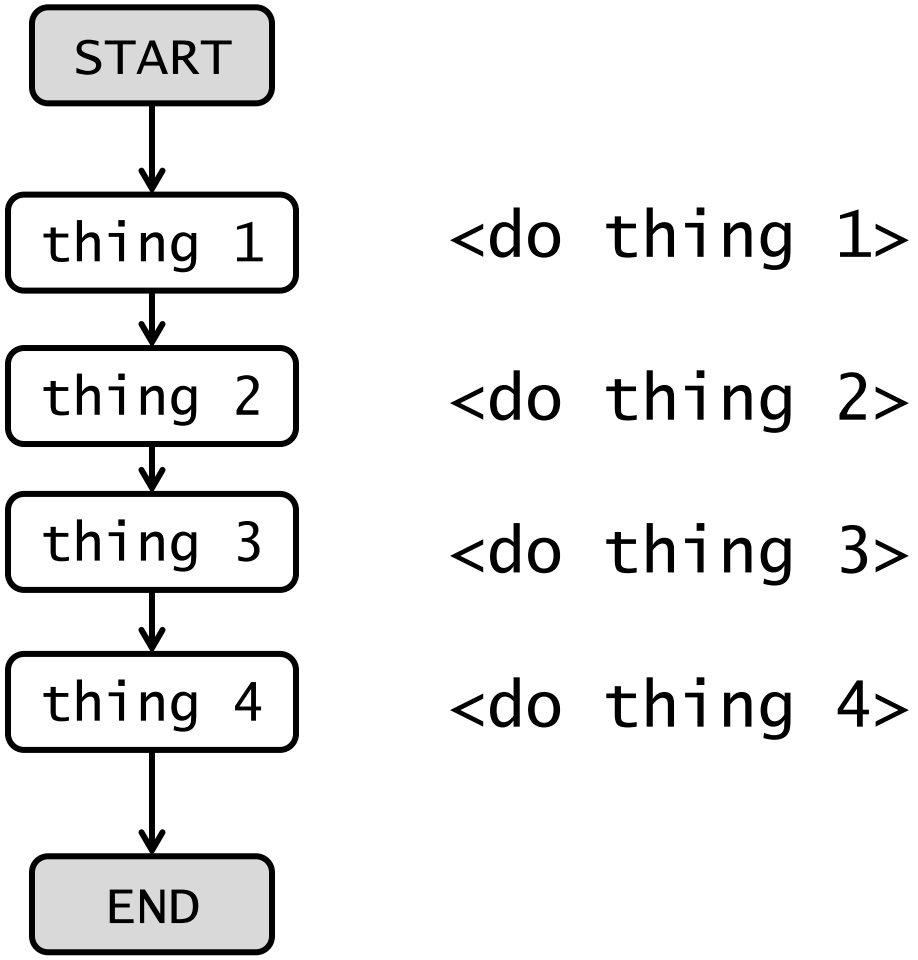
CONDITION must produce a Boolean

| | |
|---|---|
| x == y | true if x equals y |
| x ~= y | true if x does not equal y |
| x > y | true if x greater than y |
| x < y | true if x less than y |
| x >= y | true if x greater than or equal to y |
| x <= y | true if x less than or equal to y |
| x and y | true if both x and y are true |
| x or y | true if either x or y are true |
| not x | true if x is false |

# This week: for and while loops

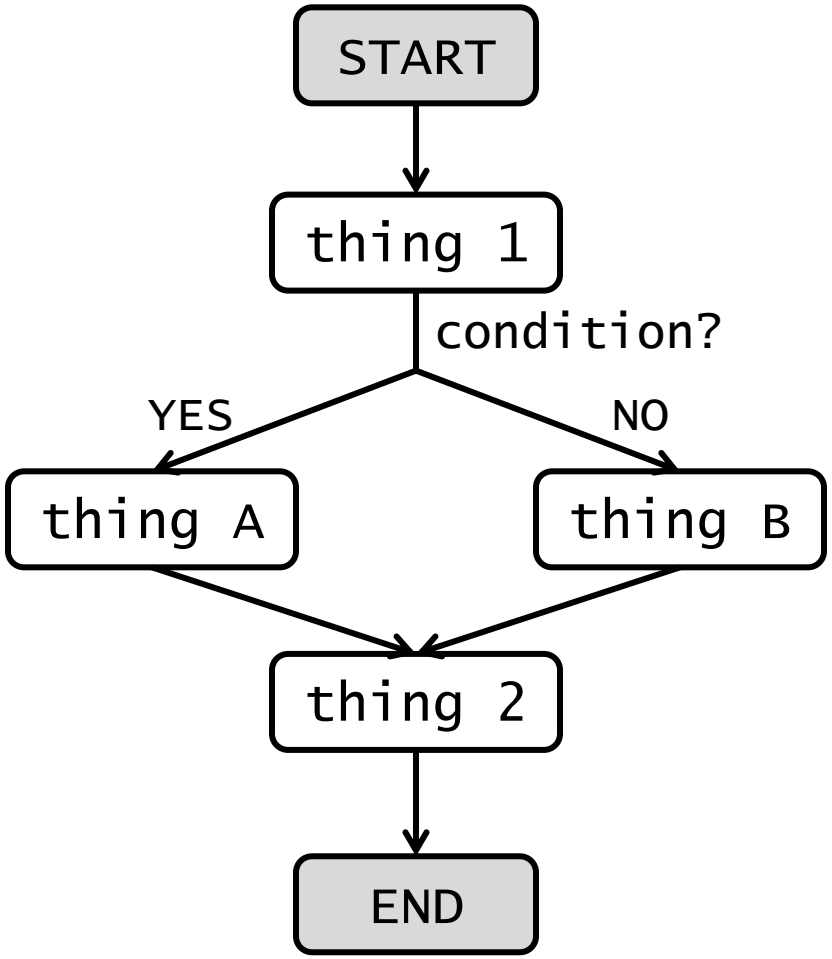
# Beyond sequential execution

Our first programs looked like this

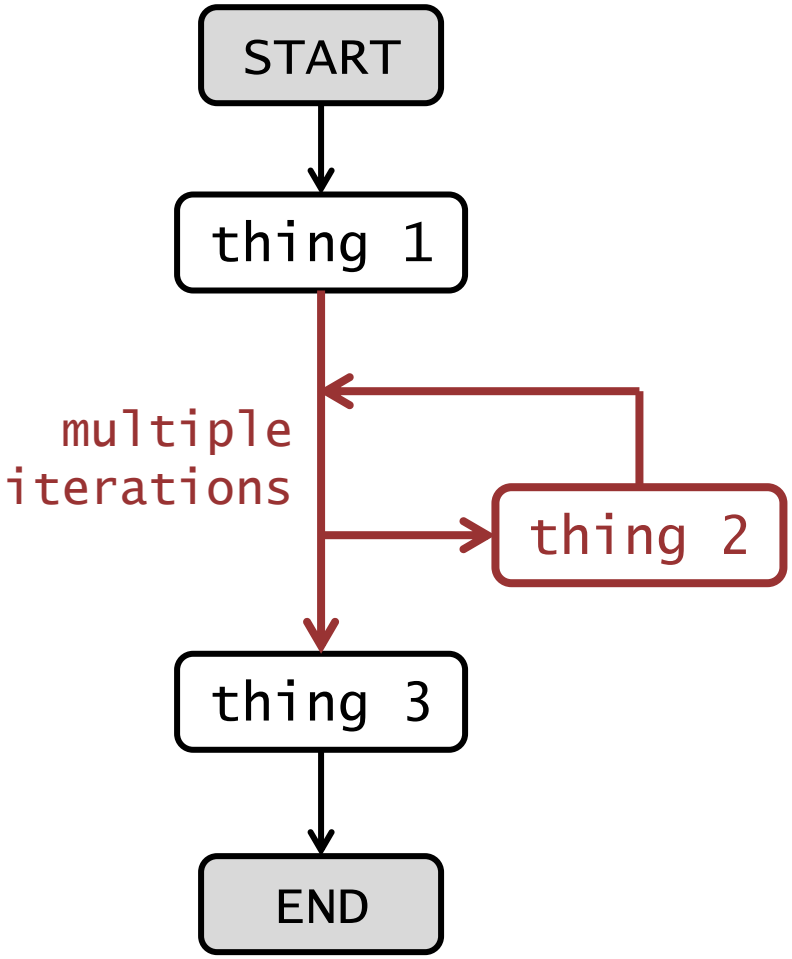


# Beyond sequential execution

Last week, sequential execution was not enough



# What if I want something more complicated?



# First: Back to Lists

```
numbers = [10, 20, 30]

list_of_lists = [[1, 2, 3], [4, 5, 6]]
```

# Getting the length of an array

**len(*s*)**

Return the length (the number of items) of an object. The argument may be a sequence (such as a string, bytes, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).

```
Example:
list1 = [1, 2, 3]
len(list1)
```

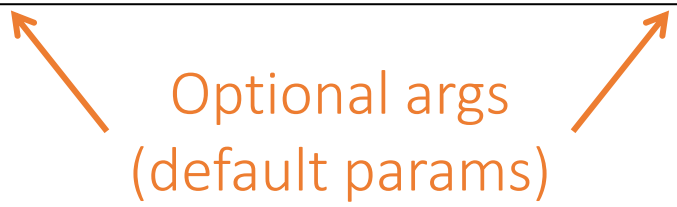
You can also use len on numpy.arrays. (We'll get to these later)

```
Example:
array1 = numpy.array([1, 2, 3])
len(array1)
```

# Using `range`

Generates a `list` from `start` up to (but excluding) `stop` in increments of `step`

```
range([start=0], stop, [step=1])
```




**Super-useful**: `range(n)`

- Generates a list from 0 up to (but excluding) n

```
range(4)
range(1,7,2)
```

# for loops

Iterate (move one by one) through the elements of a sequence (list, string, etc)

```
for ELEMENT in SEQUENCE:
    BODY
```

For each ELEMENT in SEQUENCE, the BODY is executed

BODY may or may not use ELEMENT

```
for i in [0, 1, 2, 3]:
    print('howdy \n')
```

```
for i in [0, 1, 2, 3]:
    print('i =' + str(i))
```

# Combining Loops, Conditionals, and Functions

Everything we have learned so far can be nested

```
for i in range(6) :
    if (i % 2 == 0) :
        print(str(i)+" is even")
    else:
        print(str(i)+" is odd")
```

```
for i in range(1,6) :
    for j in range(1,6) :
        print(i, j, i+j)
```

You can use commas instead of + in print as a shorthand for printing multiple values

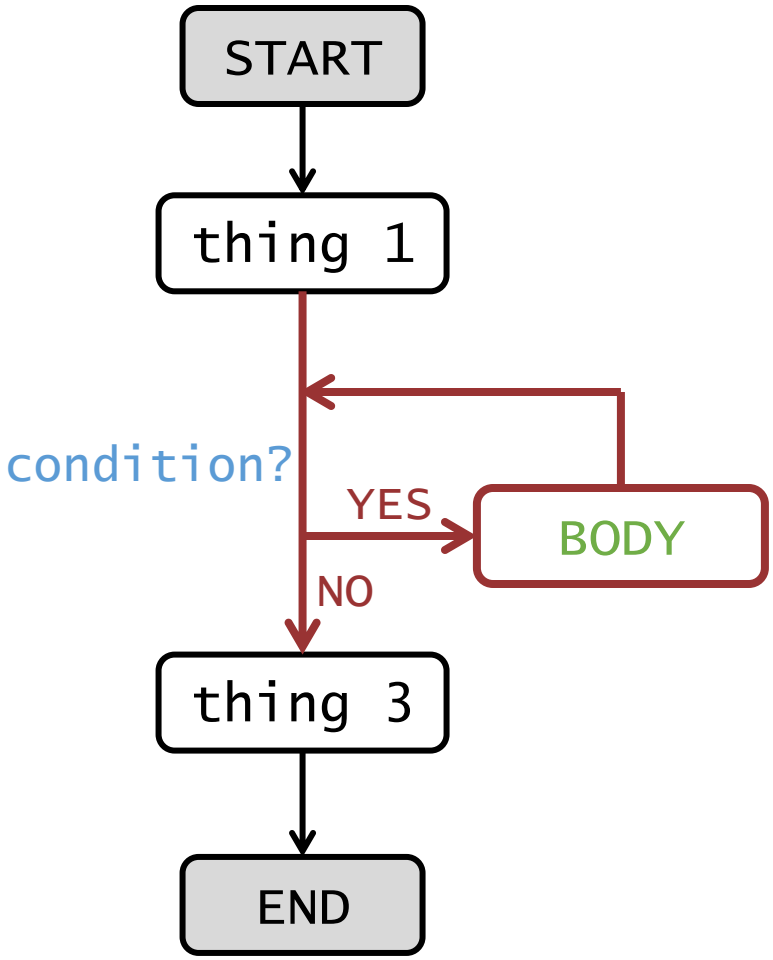
# In-class Demo Code

```
a_list = range(3,20,2)
print(list(a_list))
```

```
for i in range(6):
  if (i%2 == 0):
    print(str(i)+" is even")
  else:
    print(str(i)+" is odd")
```

```
for i in range(1,6):
    for j in range(1,6):
        print(i, j, i+j)
```

# while loops




```
while CONDITION:
    BODY
```

While CONDITION is true, BODY is executed repeatedly

BODY executed as a block

CONDITION is checked only at the start of each block execution

# Break and Continue

`break` exits out of the loop

`continue` skips the rest of the loop and continues to the next iteration

**They work on the innermost loop**

```
for i in range(5):
    for j in range(5):
        if j > i:
            break
        print(i, j)
```

# Compare

```
sum = 0
num = 0
while sum <= 100:




    sum = sum + num;
    num = num + 1

print(num)
```

```
sum = 0

for num in range(1000000):
    if sum > 100:
        break

    sum = sum + num


print(num)
```

# `for` vs `while`

`for` loop is primarily used

- for iterating over a sequence of values
- when we know the number of iterations in advance

`while` loop is primarily used

- when we don't know the number of iterations in advance (they could be controlled by user input)

# Common (semantic) errors in loops

- Forgetting to increment/decrement index variable
- Incrementing/decrementing in the wrong direction
- Nested loops: using the same index variable twice

# Some extra tricks

# Introducing: List comprehension

Compactly generate a list following a rule to assign values

```
lc_list = [EXPRESSION for ITEM in LIST]
```

Evaluate EXPRESSION on each ITEM in the LIST, appending the result to a new list

```
isodd = [bool(i%2) for i in range(10)]
print(isodd)
```

```
[False, True, False, True, False, True, False, True, False, True]
```

# Introducing: List comprehension

```
lc_list =
  [EXPRESSION for ITEM in LIST if COND]
```

Evaluate EXPRESSION on each ITEM in the LIST, appending the result to a new list, if COND is true

```
oddnums = [i for i in range(10) if (i%2)==1]
print(oddnums)
```

```
[1, 3, 5, 7, 9]
```

# In-Class: 05_ForLoops

Do this with a **different partner than last time**.

Turn in as a pair on Canvas.

Tips for pair programming:

- Switch off who is typing.
- The person who is not typing should:
  - Make comments or suggest potential solutions
  - Be “devil’s advocate”: what are potential issues with what is being typed
  - Suggest other things to explore

**At-Home:** HW 2 due tomorrow