# LECTURE 22

## MORE ON BOOLEANS AND ITERABLES

MCS 260 Fall 2021

Emily Dumas

# REMINDERS

- Homework due every Tuesday 10am
- Thursday lab students: Attempt problem 1 on the worksheet

# ASSIGNMENT WITH OPERATION

Common pattern: Modify the value of a variable using an operation.

```
x = x + 25
```

There is a shorter syntax for this:

```
x += 25
```

Not repeating the name `x` twice can help avoid errors.

| Old way | New way |
| --- | --- |
| `a = a + b` | `a += b` |
| `a = a - b` | `a -= b` |
| `a = a * b` | `a *= b` |
| `a = a / b` | `a /= b` |
| `a = a ** b` | `a **= b` |
| `a = a % b` | `a %= b` |
| `a = a // b` | `a //= b` |

# CONTINUE

This is a loose end from our discussion of loops in Lecture 6.

We talked about `break`, which exits the loop immediately.

There is also `continue`, which skips the rest of the loop body and starts the next iteration.

# WHEN TO USE CONTINUE

Never truly required. But when a loop body needs to end early, and move on to the next iteration, it can be helpful.

Using `continue` avoids most of the loop body being inside `else`.

E.g. let's modify `terminal2.py`

# NEVER NEVER NEVER

**Never** have `continue` as the last statement in a loop body. Doing so has no function—continuing at the end is the default behavior!

# NONE

`None` is the only value of type `NoneType`. It represents the absence of a value, in cases where some value is needed.

E.g. `None` is the return value of a function that doesn't have a `return` statement.

```
def f(x):
    x*x

print(f(2))
```

# BOOLEAN COERCION

Recall that every value in Python can be converted to a boolean when needed (e.g. in conditional).

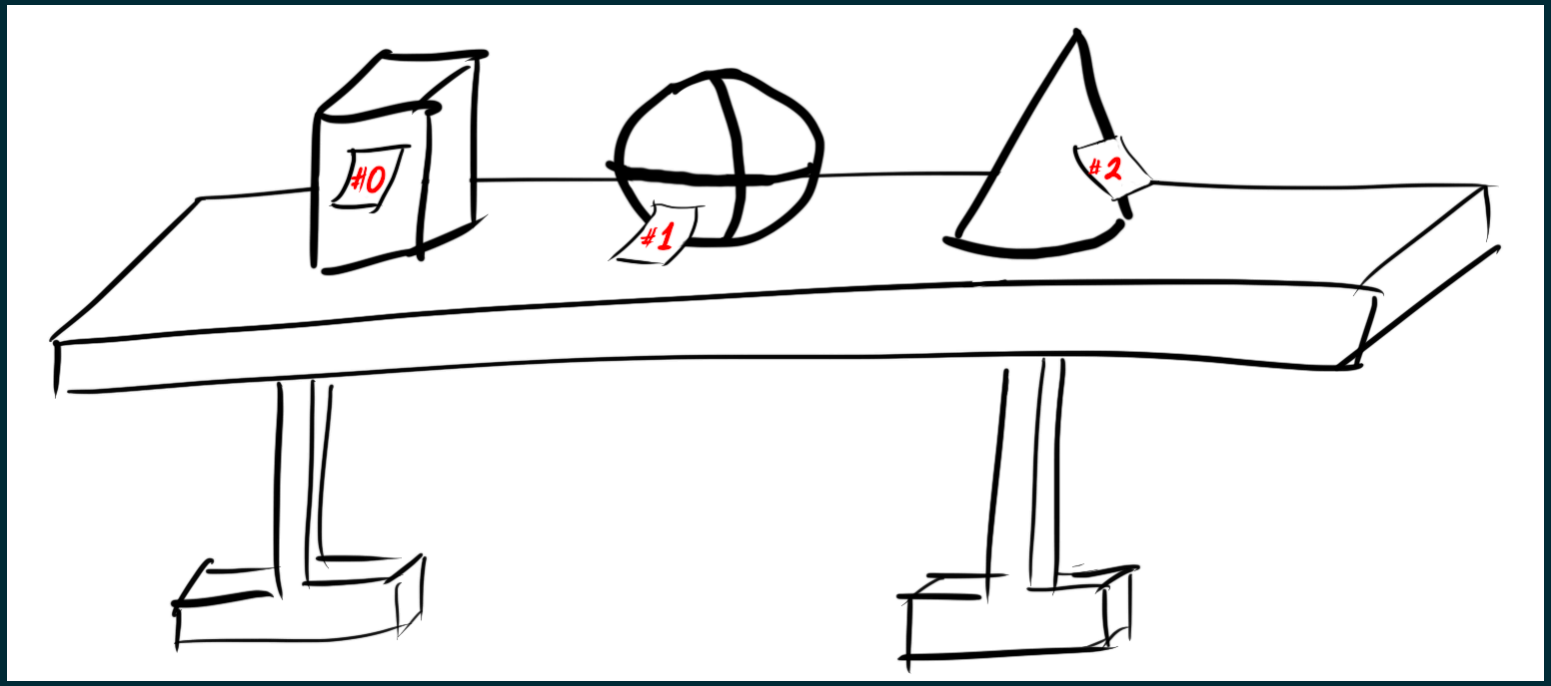
- False, None, 0, 0.0, 0j, [], {}, "" → False
- Anything else → True

Values called "truthy" or "falsy" according to what they convert to.

```
x = 5
while x:
    print(x,end=" ")
    x -= 1
```

# SEQUENCES AND ITERABLES

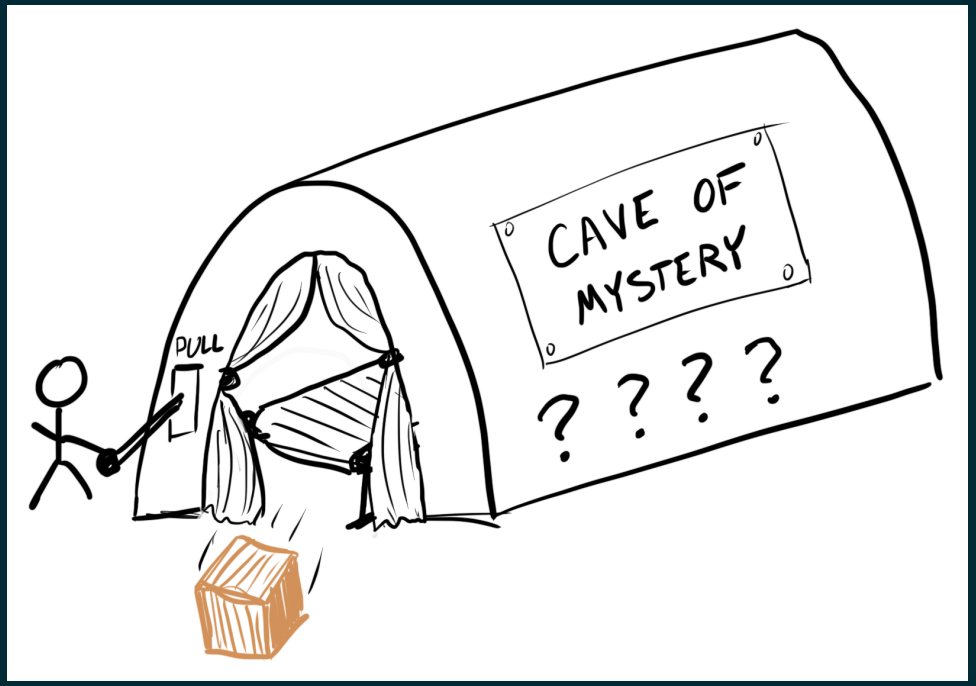
A sequence is an ordered collection that can be accessed by integer index, e.g. tuple, list, string.




Roughly: `x` is a sequence if `x[2]` might work.

# SEQUENCES AND ITERABLES

An iterable is a collection that can return items one by one upon request. (Any sequence can do this.)




`x` is an iterable if `for t in x:` works.

# NON-SEQUENCE ITERABLES

range objects, file objects,
csv.reader, dictionaries, ...

# LIST()

The function `list(x)` will convert an iterable `x` to a list, which is a sequence.

In the process it will generate and store everything the iterable produces.

Always look for a way to avoid this function. Nearly all uses by beginners are unnecessary.

# ANY & ALL

The functions `any(L)` and `all(L)` convert an iterable `L` into a single boolean.

`any(L)` returns True if at least one item from `L` is truthy. It returns as soon as it finds a truthy value. It is like a chain of `or`.

`all(L)` returns True if all items from `L` are truthy. It returns as soon as it finds a falsy value. It is like a chain of `and`.

Example: Check whether all characters in a string satisfy a condition.

```
left_keys = "qwertasdfgzxcvb"

def is_left_hand(word):
    "Can `word` be typed with only left hand on en-us keyboard?"
    return all( [c in left_keys for c in word] )
```

Example: Check whether a list of numbers contains at least one positive number.

```
def contains_a_positive(L):
    "Does `L` contain an element greater than zero?"
    return any( [x>0 for x in L] )
```

# ENUMERATE

Reminder about this useful iterable.

If `L` is an iterable giving items `a,b,c,...` then `enumerate(L)` is an iterable giving tuples `(0,a), (1,b), (2,c), ...`

# REFERENCES

- In *Downey*:
  - Section 19.4 covers `any` and `all`

# REVISION HISTORY

- 2021-10-13 Correct typos
- 2021-10-13 Initial publication