LECTURE 18

MORE ON BOOLEANS AND ITERABLES

MCS 260 Fall 2020

Emily Dumas

REMINDERS

- Quiz 6 due today
- Project 2 due Friday at 6:00pm central

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):
...     "Do nothing"
...
>>> f(1) == None
True
```

BOOL()

The built-in function `bool(x)` converts a value `x` to a boolean, i.e. to either True or False.

How? **A few values convert to False** (are "falsy"):

- False
- None
- Zero in any numeric type (0, 0.0, 0j)
- Empty containers, i.e. (), [], "", {}, range(0)

Anything else converts to True, i.e. is "truthy" (unless you use an advanced technique to override this).

AUTOMATIC BOOL CONVERSION

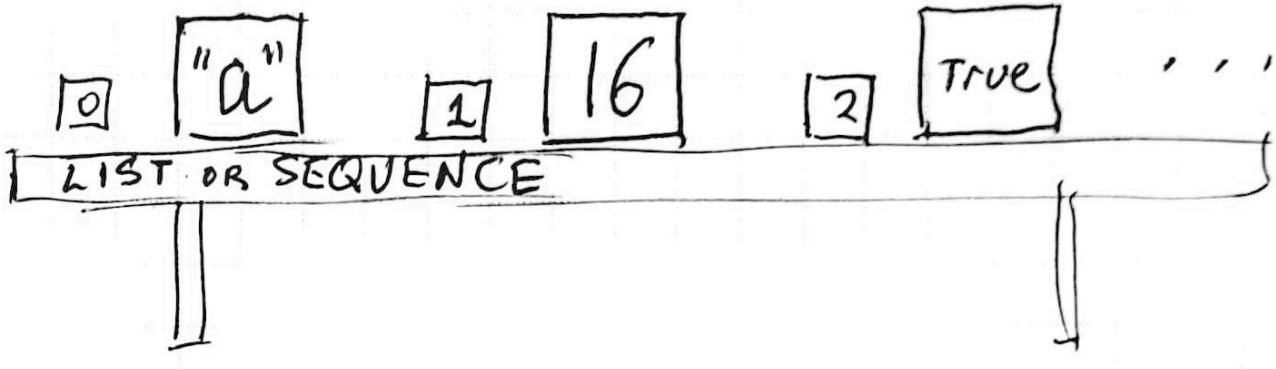
Python implicitly applies `bool()` to any value appearing where a boolean is expected, i.e. after `if`, `elif`, or `while`, or as operand of `not`, `or`, `and`.

```
>>> x = 5
>>> while x:  # not recommended; `while x!=0` is better.
...     print(x,end=" ")
...     x = x - 1
...
5 4 3 2 1 >>>
```

```
if not username:
    # Handle empty username
    print("The username must not be empty.")
    continue
```

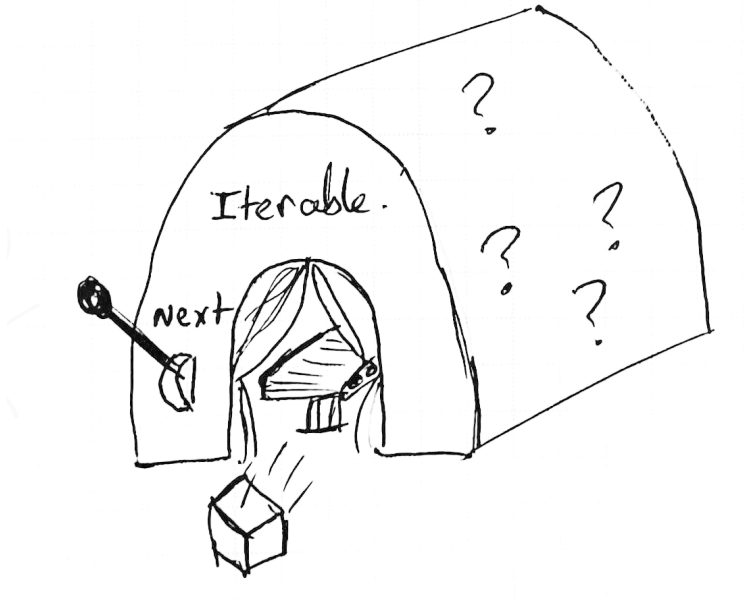
SEQUENCES AND ITERABLES

Reminder: Sequence is an ordered collection that can be accessed by integer index, e.g. tuple, list, string.



SEQUENCES AND ITERABLES

Reminder: Iterable is a collection that can return items one by one upon request, e.g. range(), dict, dict_keys, ...



ZIP

You have a list

```
xcoords = [1,2,7,0,2]
```

and another list

```
ycoords = [5,5,-1,0,1]
```

How would you make the list of *corresponding pairs*

```
[ (1,5), (2,5), (7,-1), (0,0), (2,1) ]
```

?

Could use indexing and a for loop or comprehension, e.g.

```
>>> [ (xcoords[i], ycoords[i]) for i in range(len(xcoords)) ]
[(1, 5), (2, 5), (7, -1), (0, 0), (2, 1)]
```

But remember `range(len())` usually means there is a better way?

`zip(A,B,C,...)` takes a bunch of iterables and returns tuples of values until one iterable is exhausted.

```
>>> zip(xcoords,ycoords)
<zip object at 0x7f51a3e36dc0>
>>> list(zip(xcoords,ycoords))
[(1, 5), (2, 5), (7, -1), (0, 0), (2, 1)]
```

Note `zip()` returns an iterable that we can convert to a list if needed.

`zip()` is most often used in loops

```
cols = ["name", "quiz 1", "quiz 2"]
vals = ["Anne Example", "82.5", "95.0"]
for column,value in zip(cols,vals):
    print("Found value {} in column {}".format(value,column))
```

Exercise: Given the list

```
[ 4, 8, 15, 16, 23 42 ]
```

How would you iterate over the adjacent pairs without using indices?

```
>>> for a,b in adjacent_pairs( [ 4, 8, 15, 16, 23, 42 ] ):
...     print("Pair: {} and {}".format(a,b))
...
Pair: 4 and 8
Pair: 8 and 15
Pair: 15 and 16
Pair: 16 and 23
Pair: 23 and 42
```

```
def adjacent_pairs(L):
    return zip(L,L[1:])
```

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] )
```

REFERENCES

- In *Downey*:
  - Section 19.4 covers `any` and `all`
  - Section 12.5 covers `zip`

REVISION HISTORY

- 2020-10-04 Correction about early return from all()
- 2020-10-03 Initial publication