

for loops

(and list comprehensions)

ESS 116 | Fall 2024

Prof. Henri Drake, Prof. Jane Baldwin, and Prof. Michael Pritchard

(Modified from Ethan Campbell and Katy Christensen's materials for UW's Ocean 215)

What we'll cover in this lesson

1. **for loops**
2. Advanced `for` loops
3. List comprehensions

Why use loops when programming?

Sometimes we want to do something repetitive:

```
1 temperatures = [68.5, 72.1, 74.8, 65.3, 62.7, 58.9]
2 print(temperatures[0])
3 print(temperatures[1])
4 print(temperatures[2])
5 print(temperatures[3])
6 print(temperatures[4])
7 print(temperatures[5])
```

```
☞ 68.5
   72.1
   74.8
   65.3
   62.7
   58.9
```

Why use loops when programming?

**Sometimes we want to
do something repetitive:**

```
1 print(temperatures[0])
2 print(temperatures[1])
3 print(temperatures[2])
4 print(temperatures[3])
5 print(temperatures[4])
6 print(temperatures[5])
7 print(temperatures[6])
8 print(temperatures[7])
9 print(temperatures[8])
10 print(temperatures[9])
11 print(temperatures[10])
12 print(temperatures[11])
13 print(temperatures[12])
14 print(temperatures[13])
15 print(temperatures[14])
16 print(temperatures[15])
17 print(temperatures[16])
18 print(temperatures[17])
19 print(temperatures[18])
20 print(temperatures[19])
21 print(temperatures[20])
22 print(temperatures[21])
23 print(temperatures[22])
24 print(temperatures[23])
25 print(temperatures[24])
26 print(temperatures[25])
27 print(temperatures[26])
28 print(temperatures[27])
29 print(temperatures[28])
30 print(temperatures[29])
31 print(temperatures[30])
32 print(temperatures[31])
33 print(temperatures[32])
34 print(temperatures[33])
35 print(temperatures[34])
36 print(temperatures[35])
37 print(temperatures[36])
38 print(temperatures[37])
39 print(temperatures[38])
40 print(temperatures[39])
41 print(temperatures[40])
42 print(temperatures[41])
43 print(temperatures[42])
44 print(temperatures[43])
45 print(temperatures[44])
46 print(temperatures[45])
47 print(temperatures[46])
48 print(temperatures[47])
49 print(temperatures[48])
50 print(temperatures[49])
```

```
51 print(temperatures[50])
52 print(temperatures[51])
53 print(temperatures[52])
54 print(temperatures[53])
55 print(temperatures[54])
56 print(temperatures[55])
57 print(temperatures[56])
58 print(temperatures[57])
59 print(temperatures[58])
60 print(temperatures[59])
61 print(temperatures[60])
62 print(temperatures[61])
63 print(temperatures[62])
64 print(temperatures[63])
65 print(temperatures[64])
66 print(temperatures[65])
67 print(temperatures[66])
68 print(temperatures[67])
69 print(temperatures[68])
70 print(temperatures[69])
71 print(temperatures[70])
72 print(temperatures[71])
73 print(temperatures[72])
74 print(temperatures[73])
75 print(temperatures[74])
76 print(temperatures[75])
77 print(temperatures[76])
78 print(temperatures[77])
79 print(temperatures[78])
80 print(temperatures[79])
81 print(temperatures[80])
82 print(temperatures[81])
83 print(temperatures[82])
84 print(temperatures[83])
85 print(temperatures[84])
86 print(temperatures[85])
87 print(temperatures[86])
88 print(temperatures[87])
89 print(temperatures[88])
90 print(temperatures[89])
91 print(temperatures[90])
92 print(temperatures[91])
93 print(temperatures[92])
94 print(temperatures[93])
95 print(temperatures[94])
96 print(temperatures[95])
97 print(temperatures[96])
98 print(temperatures[97])
99 print(temperatures[98])
100 print(temperatures[99])
```


Why use loops when programming?

Sometimes we want to
do something repetitive:

```
1 print(temperatures[0])
2 print(temperatures[1])
3 print(temperatures[2])
4 print(temperatures[3])
5 print(temperatures[4])
6 print(temperatures[5])
7 print(temperatures[6])
8 print(temperatures[7])
9 print(temperatures[8])
10 print(temperatures[9])
11 print(temperatures[10])
12 print(temperatures[11])
13 print(temperatures[12])
14 print(temperatures[13])
15 print(temperatures[14])
16 print(temperatures[15])
17 print(temperatures[16])
18 print(temperatures[17])
19 print(temperatures[18])
20 print(temperatures[19])
21 print(temperatures[20])
22 print(temperatures[21])
23 print(temperatures[22])
24 print(temperatures[23])
25 print(temperatures[24])
26 print(temperatures[25])
27 print(temperatures[26])
28 print(temperatures[27])
29 print(temperatures[28])
30 print(temperatures[29])
31 print(temperatures[30])
32 print(temperatures[31])
33 print(temperatures[32])
34 print(temperatures[33])
35 print(temperatures[34])
36 print(temperatures[35])
37 print(temperatures[36])
38 print(temperatures[37])
39 print(temperatures[38])
40 print(temperatures[39])
41 print(temperatures[40])
42 print(temperatures[41])
43 print(temperatures[42])
44 print(temperatures[43])
45 print(temperatures[44])
46 print(temperatures[45])
47 print(temperatures[46])
48 print(temperatures[47])
49 print(temperatures[48])
50 print(temperatures[49])
```

Oops.

```
51 print(temperatures[50])
52 print(temperatures[51])
53 print(temperatures[52])
54 print(temperatures[53])
55 print(temperatures[54])
56 print(temperatures[55])
57 print(temperatures[56])
58 print(temperatures[57])
59 print(temperatures[58])
60 print(temperatures[59])
61 print(temperatures[60])
62 print(temperatures[61])
63 print(temperatures[62])
64 print(temperatures[63])
65 print(temperatures[64])
66 print(temperatures[65])
67 print(temperatures[66])
68 print(temperatures[67])
69 print(temperatures[68])
70 print(temperatures[69])
71 print(temperatures[70])
72 print(temperatures[71])
73 print(temperatures[72])
74 print(temperatures[73])
75 print(temperatures[74])
76 print(temperatures[75])
77 print(temperatures[76])
78 print(temperatures[77])
79 print(temperatures[78])
80 print(temperatures[79])
81 print(temperatures[80])
82 print(temperatures[81])
83 print(temperatures[82])
84 print(temperatures[83])
85 print(temperatures[84])
86 print(temperatures[85])
87 print(temperatures[86])
88 print(temperatures[87])
89 print(temperatures[88])
90 print(temperatures[89])
91 print(temperatures[90])
92 print(temperatures[91])
93 print(temperatures[92])
94 print(temperatures[93])
95 print(temperatures[94])
96 print(temperatures[95])
97 print(temperatures[96])
98 print(temperatures[97])
99 print(temperatures[98])
100 print(temperatures[99])
```

Why use loops when programming?

Sometimes we want to
do so

```
1 print(temperatures[0])
2 print(temperatures[1])
3 print(temperatures[2])
4 print(temperatures[3])
5 print(temperatures[4])
6 print(temperatures[5])
7 print(temperatures[6])
8 print(temperatures[7])
```

```
51 print(temperatures[50])
52 print(temperatures[51])
53 print(temperatures[52])
54 print(temperatures[53])
55 print(temperatures[54])
56 print(temperatures[55])
57 print(temperatures[56])
58 print(temperatures[57])
```

There's a more efficient way.
Loops allow you to repeat an
action, *efficiently*.

```
35 print(temperatures[34])
36 print(temperatures[35])
37 print(temperatures[36])
38 print(temperatures[37])
39 print(temperatures[38])
40 print(temperatures[39])
41 print(temperatures[40])
42 print(temperatures[41])
43 print(temperatures[42])
44 print(temperatures[43])
45 print(temperatures[44])
46 print(temperatures[45])
47 print(temperatures[46])
48 print(temperatures[47])
49 print(temperatures[48])
50 print(temperatures[49])
```

```
85 print(temperatures[84])
86 print(temperatures[85])
87 print(temperatures[86])
88 print(temperatures[87])
89 print(temperatures[88])
90 print(temperatures[89])
91 print(temperatures[90])
92 print(temperatures[91])
93 print(temperatures[92])
94 print(temperatures[93])
95 print(temperatures[94])
96 print(temperatures[95])
97 print(temperatures[96])
98 print(temperatures[97])
99 print(temperatures[98])
100 print(temperatures[99])
```

Elements of the Python for loop

for *<VARIABLE>* **in** *<ITERABLE>*:

<ACTION>

<ACTION>

etc.

Elements of the Python for loop

You should give this variable a unique name

In Python, iterables are collections of objects

for *<VARIABLE>* **in** *<ITERABLE>*:

<ACTION>

<ACTION>

etc.

The colon is essential!

Actions that you want to repeat can be any line of code, such as `print` statements, variable assignments, or calculations

Indent using a **tab** or **2 spaces** (on Google Colab)

Iterables that you can use in for loops

list

`[4, 3, 2, 1]`

tuple

`('pH', 'puget_sound', 7.8)`

string

`'hello'`

range()

`range(0, 7, 2)`

enumerate()

stay tuned...

zip()

stay tuned...

and others...

Iterables that you can use in for loops

list

`[4, 3, 2, 1]`

tuple

`('pH', 'puget_sound', 7.8)`

string

`'hello'`

range()

`range(0, 7, 2)` a.k.a. `[0, 2, 4, 6]`

enumerate()

stay tuned...

zip()

stay tuned...

and others...

Iterables that you can use in for loops

Variable names are okay to use in loops, too:

list

tuple

string

range()

enumerate()

zip()

and others...

countdown

pH_data

hello_string

even_numbers

stay tuned...

stay tuned...

Examples of for loops

Option 1:

```
1 for item in [4,3,2,1]:  
2     print(item)
```

☞ 4
3
2
1

Option 2:

```
1 countdown = [4,3,2,1]  
2  
3 for item in countdown:  
4     print(item)
```

☞ 4
3
2
1

Examples of for loops

Option 1:

```
1 for value in ('pH', 'puget_sound', 7.8):  
2     print(value)
```

```
☞ pH  
   puget_sound  
   7.8
```

Option 2:

```
1 pH_data = ('pH', 'puget_sound', 7.8)  
2  
3 for value in pH_data:  
4     print(value)
```

```
☞ pH  
   puget_sound  
   7.8
```

Examples of for loops

Option 1:

```
1 for character in 'hello':  
2     print(character)
```

```
☞ h  
   e  
   l  
   l  
   o
```

Option 2:

```
1 hello_string = 'hello'  
2  
3 for character in hello_string:  
4     print(character)
```

```
☞ h  
   e  
   l  
   l  
   o
```

Examples of for loops

Option 1:

```
1 for index in range(0,7,2):  
2     print(index)
```

☞ 0
2
4
6

Option 2:

```
1 even_numbers = range(0,7,2)  
2  
3 for index in even_numbers:  
4     print(index)
```

☞ 0
2
4
6

Using a for loop to calculate a sum of numbers

Option 1:

```
1 numbers = [5,6,7,8]
2 sum = 0
3
4 for value in numbers:
5     sum = sum + value
6
7 print('The sum is:',sum)
```

☞ The sum is: 26

Using a for loop to calculate a sum of numbers

Option 2:

```
1 numbers = [5,6,7,8]
2 sum = 0
3
4 for value in numbers:
5     sum += value
6
7 print('The sum is:', sum)
```

Assignment operator:


$a += b$
is equivalent to:
 $a = a + b$

☞ The sum is: 26

Using a for loop to calculate a sum of numbers

Option 3:

```
1 numbers = [5,6,7,8]
2 sum = 0
3
4 for index in range(len(numbers)):
5     sum += numbers[index]
6
7 print('The sum is:', sum)
```

 **range(4) a.k.a. [0, 1, 2, 3]**

☞ The sum is: 26

What we'll cover in this lesson

1. `for` loops
- 2. Advanced `for` loops**
3. List comprehensions

Nested for loops

```
1 params = ['Temperature', 'Salinity', 'Oxygen']
2 units = ['°C', 'PSU', 'μmol/kg']
3 currents_mix = [[4.4, 4.8, 4.5],      # temp (°C)
4                  [34.5, 33.9, 33.8], # salinity (PSU)
5                  [230, 250, 260]]    # oxygen (μmol/kg)
6
7 n_params = len(currents_mix)          # 3 parameters
8 n_currents = len(currents_mix[0])    # 3 currents
```

Mix seawater from 3 locations.

What is the average temperature, salinity, and oxygen?

Nested for loops

```
1 params = ['Temperature', 'Salinity', 'Oxygen']
2 units = ['°C', 'PSU', 'μmol/kg']
3 currents_mix = [[4.4, 4.8, 4.5],      # temp (°C)
4                  [34.5, 33.9, 33.8], # salinity (PSU)
5                  [230, 250, 260]]    # oxygen (μmol/kg)
6
7 n_params = len(currents_mix)          # 3 parameters
8 n_currents = len(currents_mix[0])    # 3 currents
9
10 for param_idx in range(n_params):    ← Outer for loop
11     sum = 0.0
12
13     for current_idx in range(n_currents): ← Inner for loop
14         sum += currents_mix[param_idx][current_idx]
15
16     average_val = sum / n_currents
17     print(params[param_idx] + ' (' + units[param_idx] + '):', average_val)
```

Indent!

Nested for loops

Cycle	Outer loop's param_idx	Inner loop's current_idx	currents_mix[param_idx][current_idx]	
#1	0	0	4 . 4	Temperature
#2	0	1	4 . 8	
#3	0	2	4 . 5	
#4	1	0	34 . 5	Salinity
#5	1	1	33 . 9	
#6	1	2	33 . 8	
#7	2	0	230	Oxygen
#8	2	1	250	
#9	2	2	260	

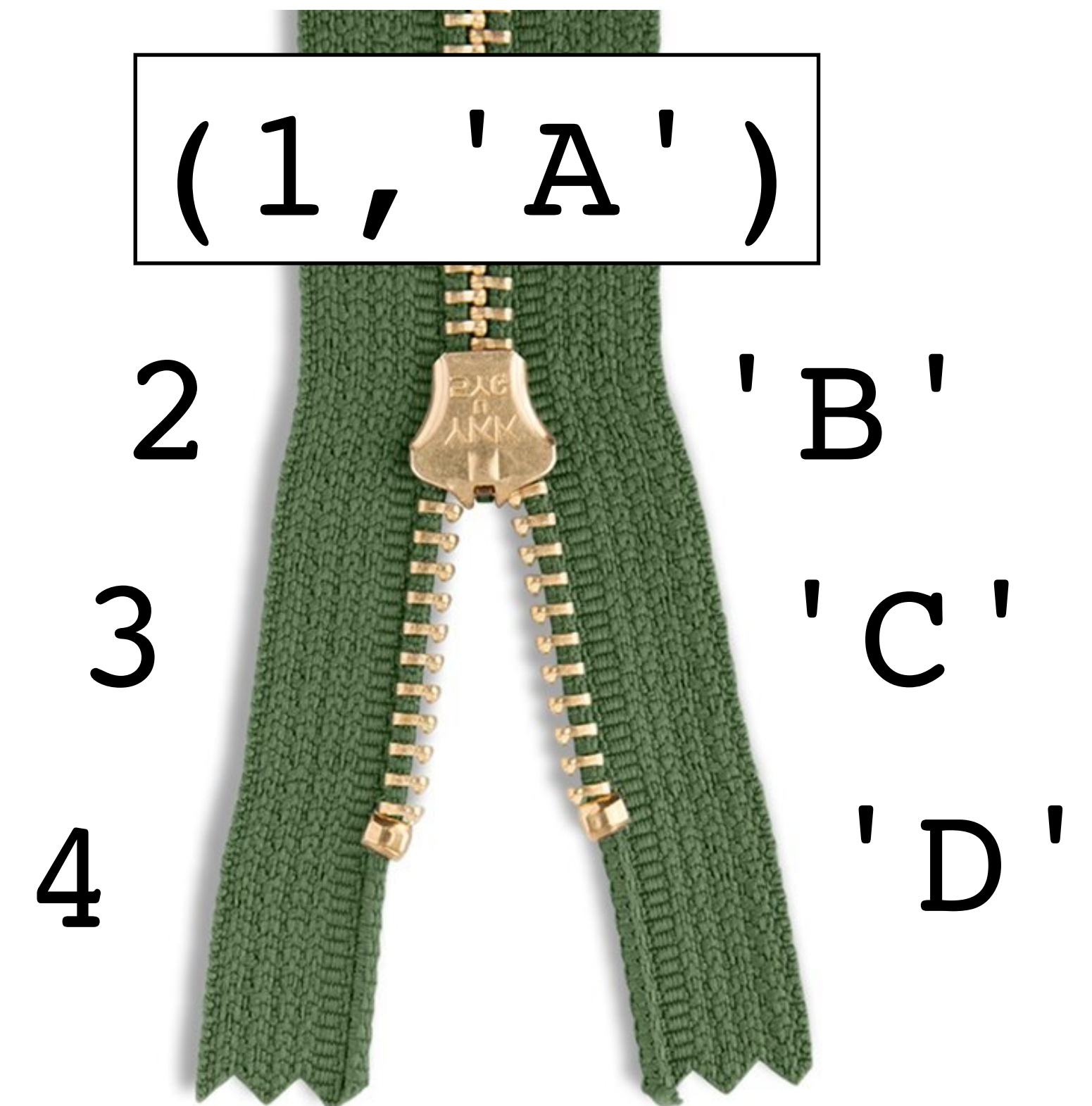
Nested for loops

Cycle	Outer loop's param_idx	Inner loop's current_idx	currents_mix[param_idx][current_idx]
#1	0	0	4.4
#2	0	1	4.8
<div>↳ Temperature (°C): 4.5666666666666666 Salinity (PSU): 34.066666666666667 Oxygen (μmol/kg): 246.66666666666666</div>			
#6	1	2	33.8
#7	2	0	230
#8	2	1	250
#9	2	2	260

Looping using the `zip()` function

`zip()` joins multiple iterators (e.g. lists) and returns an iterable of tuples.
Those tuples get unpacked when looping over the `zip` object.

```
1 x = [1,2,3,4]
2 y = ['A','B','C','D']
3
4 zip(x,y)
```



Looping using the `zip()` function

`zip()` joins multiple iterators (e.g. lists) and returns an iterable of tuples. Those tuples get unpacked when looping over the `zip` object.

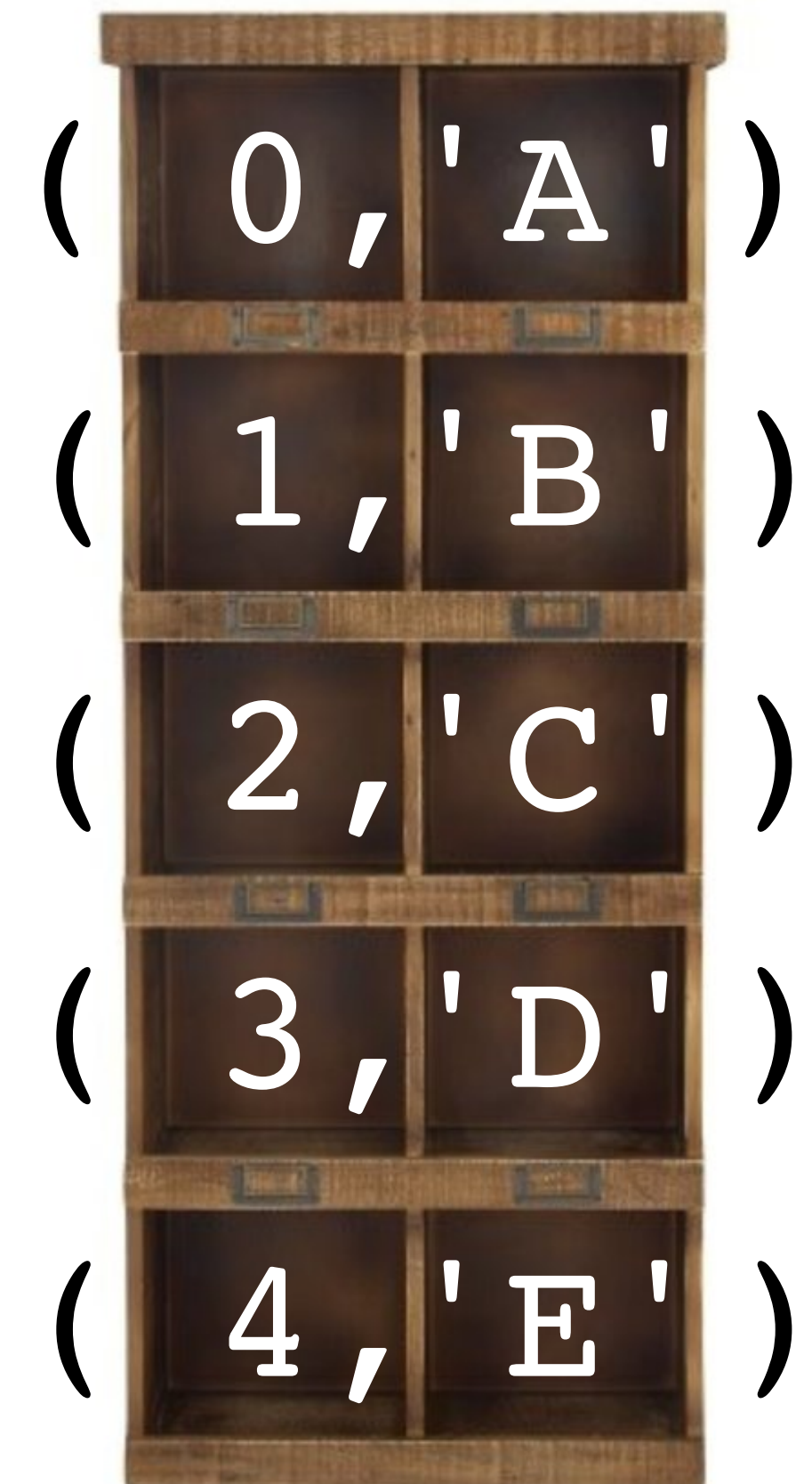
```
1 params = ['Temperature', 'Salinity', 'Oxygen']
2 units = ['°C', 'PSU', 'μmol/kg']
3
4 for param, unit in zip(params, units):
5     print(param, 'has units of', unit)
```

```
↳ Temperature has units of °C
   Salinity has units of PSU
   Oxygen has units of μmol/kg
```

Looping using the `enumerate()` function

`enumerate()` takes an iterable (e.g. a list) as an argument and returns an iterable of tuple pairs of (`index`, `value`). Index starts counting from 0.

```
1 x = ['A', 'B', 'C', 'D', 'E']  
2  
3 enumerate(x)
```



Looping using the `enumerate()` function

`enumerate()` takes an iterable (e.g. a list) as an argument and returns an iterable of tuple pairs of `(index, value)`. Index starts counting from 0.

```
1 abbrevs = ['POC', 'DOC', 'DIC']
2 names = ['particulate organic carbon',
3          'dissolved organic carbon',
4          'dissolved inorganic carbon']
5
6 for index, abbrev in enumerate(abbrevs):
7     print(abbrev, 'stands for', names[index])
```

```
[> POC stands for particulate organic carbon
   DOC stands for dissolved organic carbon
   DIC stands for dissolved inorganic carbon
```

What we'll cover in this lesson

1. `for` loops
2. Advanced `for` loops
- 3. List comprehensions**

An alternative to loops: list comprehensions

Create a list containing the first ten perfect squares (0^2 , 1^2 , 2^2 , 3^2 , 4^2 , etc.):

Option 1
(for loop):

```
1 squares = []  
2 for num in range(10):  
3     squares.append(num * num)  
4  
5 print(squares)
```

☞ [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

An alternative to loops: list comprehensions

Create a list containing the first ten perfect squares (0^2 , 1^2 , 2^2 , 3^2 , 4^2 , etc.):

Option 2
(list comprehension):

```
1 squares = [num * num for num in range(10)]
2
3 print(squares)
```

Calculation

This looks like a for loop!

☞ [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]