Lists

- A list is an **ordered** collection of values. To create a new list from scratch, we use [square brackets].

In [5]:
```
mixed_list = [-2, 2.5, 'michigan', [1, 3], max] # Different types!
mixed_list
```

Out[5]: `[-2, 2.5, 'michigan', [1, 3], <function max>]`

- There are a variety of built-in functions that work with lists.

In [6]:
```
max(['hey', 'hi', 'hello'])
```

Out[6]: `'hi'`

Appending

- We use the `append` **method** to add elements to the end of a list.

  Since this is a method, we call it using "dot" notation, i.e. `groceries.append(...)` instead of `append(groceries, ...)`.

In [7]:
```
1 groceries = ['eggs', 'milk']
2 groceries
```

Out[7]: ['eggs', 'milk']

In [8]:
```
1 max(groceries)
```

Out[8]: 'milk'

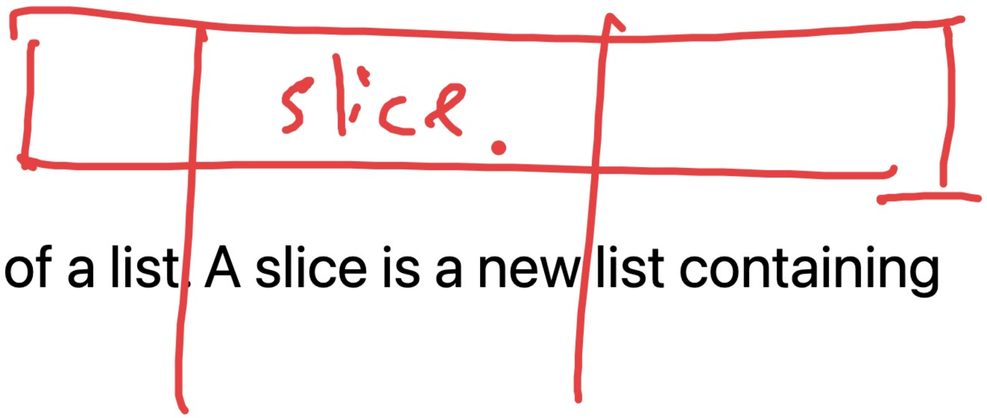
In [9]:
```
1 groceries.append('bread')
```

In [ ]:
```
1 groceries
```

max function

oppend method.

Slicing



- We can use indexes to create a "slice" of a list. A slice is a new list containing elements from another list.

In [21]:
```
1 nums
```

Out[21]: `[3, 1, 'dog', -9.5, 'michigan']`

In [ ]:
```
1 nums[1:3]
```

Slicing

- We can use indexes to create a "slice" of a list. A slice is a new list containing elements from another list.

```
In [21]: 1 nums
```

0 1 2 3

```
Out[21]: [3, 1, 'dog', -9.5, 'michigan']
```

```
In [22]: 1 nums[1:3]
```

```
Out[22]: [1, 'dog']
```

- If an object is mutable, then any name referring to it will see those changes reflected. **Be careful!**

- **Example 1**: What is the value of `y` after running the following cell?

In [53]:
```
x = 42
y = x
x = 12
y
```

Out[53]: 42

- **Example 2**: What is the value of `b` after running the following cell?

In [54]:
```
a = [5, 10]
b = a
a[0] = -1
b
```

Out[54]: [-1, 10]

Activity

Suppose we run the cell below.

```
total = 3
def square_and_cube(a, b):
    return a ** 2 + total ** b
```

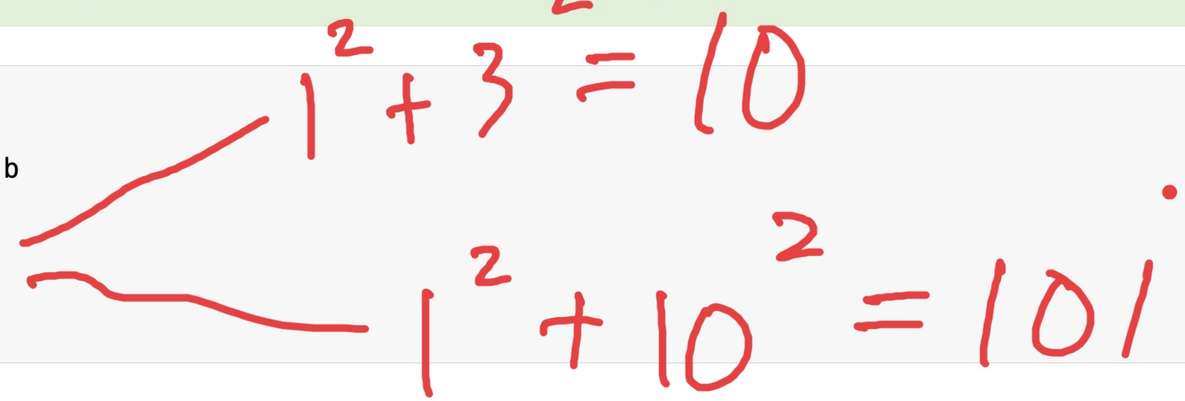
Then, suppose we run the cell below twice.

```
total = square_and_cube(1, 2)
```

What is the value of `total`? **Try and answer without writing any code.**

In [68]:
```
1 total = 3
2 def square_and_cube(a, b):
3     return a ** 2 + total ** b
4
5 total = square_and_cube(1, 2)
6 total = square_and_cube(1, 2)
7 total
```

Out[68]: 101



```
# Cell B
mystery(creature)
creature
```

Try and answer without writing any code.

In [69]:
```
def mystery(vals):
    vals[-1] = 15
    return vals.append('BBB')
```

no output, like return None.

In [79]:
```
creature = [1, 2, 3]
mystery(creature)
```

no output!

In [80]:
```
creature
```

Out[80]:
```
[1, 2, 15, 'BBB']
```

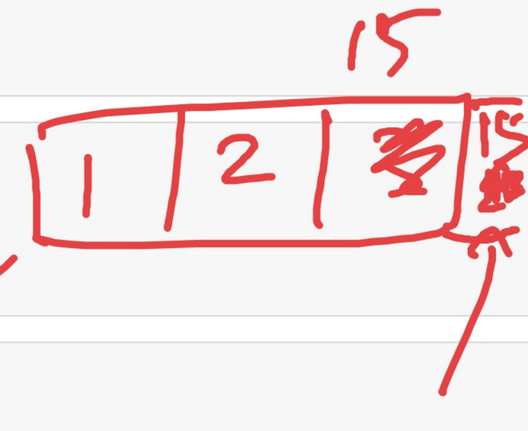
In [ ]:
```

```

In [69]:
```
def mystery(vals):
    vals[-1] = 15
    return vals.append('BBB')
```

In [79]:
```
creature = [1, 2, 3]
mystery(creature)
```

In [80]:
```
creature
```



Out[80]: [1, 2, 15, 'BBB']

In [81]:
```
creature = [1, 2, 3]
```

In [84]:
```
mystery(creature)
```

In [85]:
```
creature
```

Out[85]: [1, 2, 15, 15, 15, 'BBB']

In [122]:
```
states_dict = {}
for code in codes_dict:
    state = codes_dict[code]

    # Checks if key (not) in dictionary
    if state not in states_dict:
        states_dict[state] = []

    states_dict[state].append(code)
```

In [123]:
```
states_dict
```

Out[123]:
```
{'New Jersey': [201, 551, 609, 732, 848, 856, 862, 908, 973],
 'District of Columbia': [202],
 'Connecticut': [203, 475, 860, 959],
 'Alabama': [205, 251, 256, 334],
 'Washington': [206, 253, 360, 425, 509, 564],
 'Maine': [207],
 'Idaho': [208],
 'California': [209,
  213,
  310,
```