Week 1, Day 4: Thursday

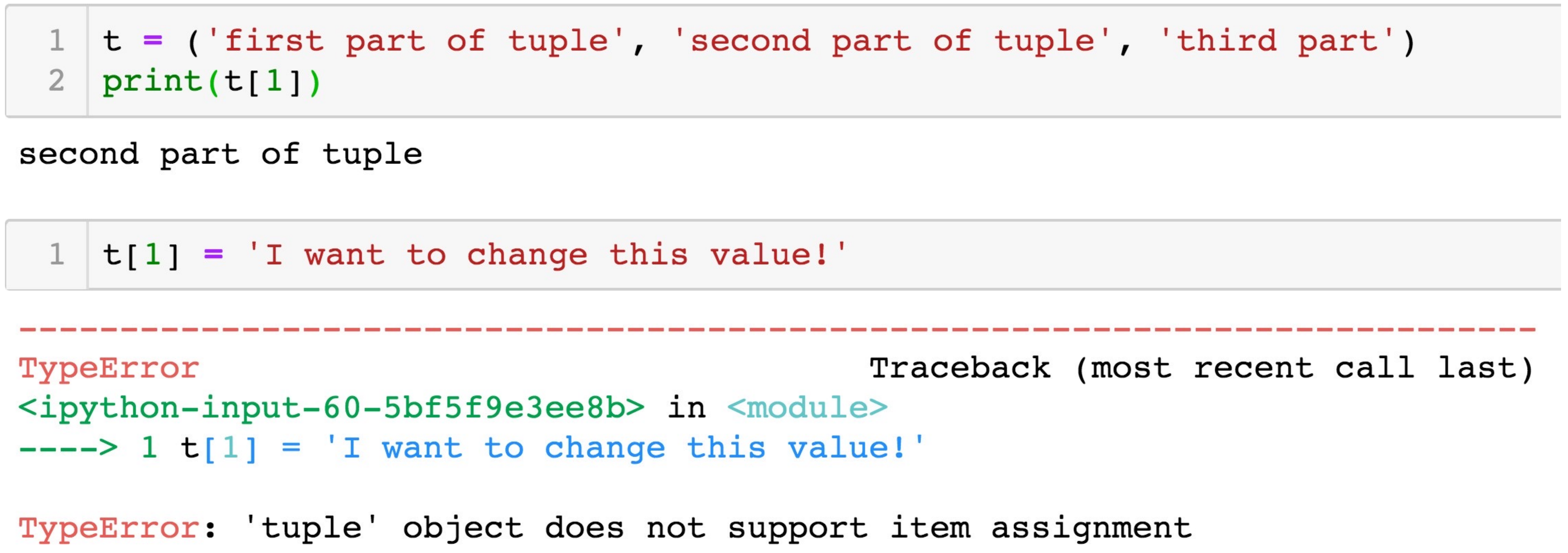
Agenda:

- Review HW
- Review Astrochemistry activity
- Tuples, sets, dictionaries
- Working with real data: reading/writing to files
- If time: algorithmic thinking with sorting lists
- If time: brief intro to recursive functions

Homework Review

Tuples

- Tuples are just like lists, except that:
  - They're denoted with parentheses: ()
  - They are immutable



```
t = ('first part of tuple', 'second part of tuple', 'third part')
print(t[1])
```

```
second part of tuple
```

```
t[1] = 'I want to change this value!'
```

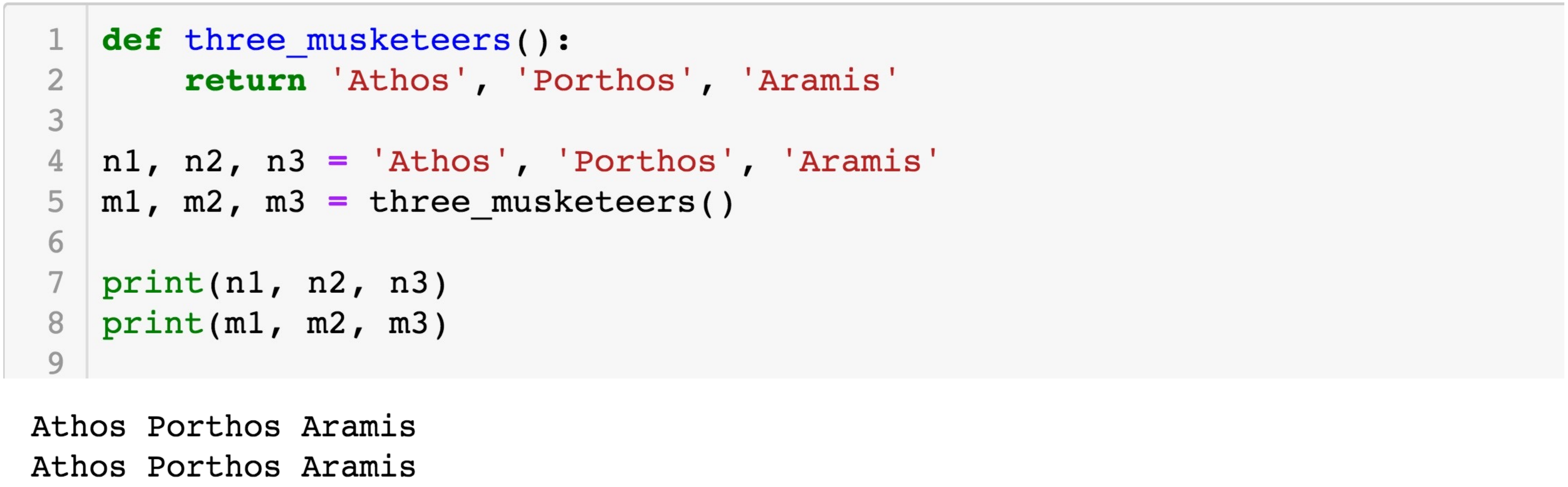
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-60-5bf5f9e3ee8b> in <module>
----> 1 t[1] = 'I want to change this value!'

TypeError: 'tuple' object does not support item assignment
```

Why use tuples instead of lists?

- Programs run a bit faster when manipulating tuples rather than lists, but this won’t be noticeable for short tuples / lists

- Sometimes you don’t want certain data to be modified. Putting it in a tuple will protect against accidental modification.

Tuple packing and unpacking

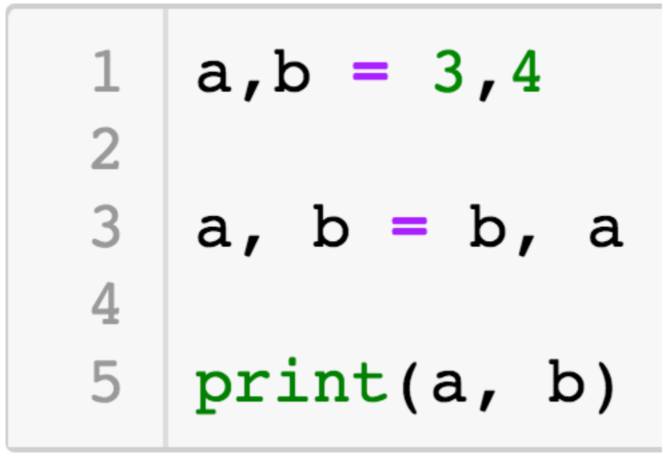


```python
def three_musketeers():
    return 'Athos', 'Porthos', 'Aramis'

n1, n2, n3 = 'Athos', 'Porthos', 'Aramis'
m1, m2, m3 = three_musketeers()

print(n1, n2, n3)
print(m1, m2, m3)

```

```
Athos Porthos Aramis
Athos Porthos Aramis
```

Number of variables on left hand side must match number of variables on right hand side!

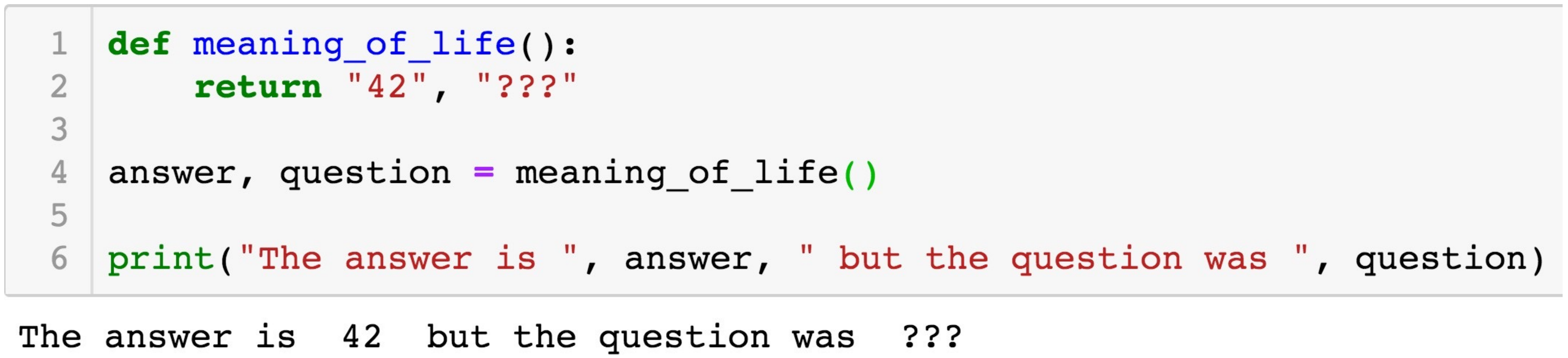
Tuples



```python
a,b = 3,4

a, b = b, a

print(a, b)
```

```
4 3
```

Returning tuples from functions



```python
def meaning_of_life():
    return "42", "???"

answer, question = meaning_of_life()

print("The answer is ", answer, " but the question was ", question)
```

```
The answer is  42  but the question was  ???
```

Activity 2: Tuples

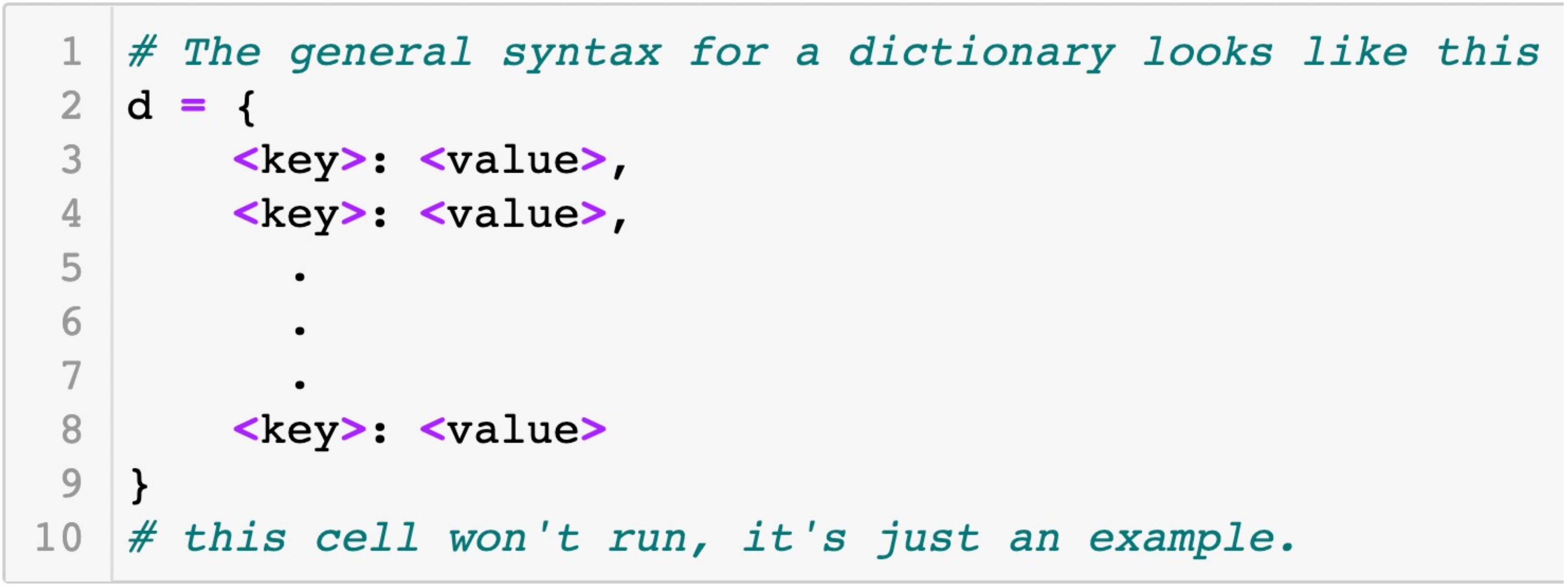
Sets

- A set is a collection of unique, unordered objects. ***Syntax:*** enclosed in braces {}
- The set() function takes as input any iterable (such as a list) and returns a set containing all the unique elements of the iterable
  - In other words, set([1, 1, 2, 1, 3]) would return {1, 2, 3}.
  - set([“Hello”, “Hi”, “Hello”, “Bonjour”]) would return {“Bonjour”, “Hello”, “Hi”}

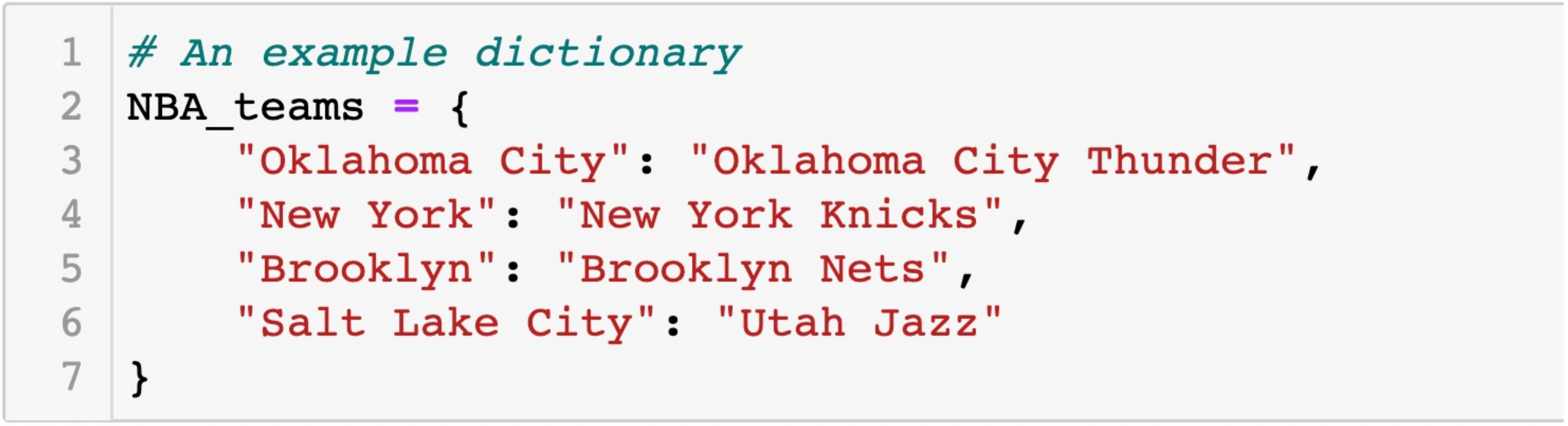
Dictionaries

- Collection of objects stored as *key-value pairs*
- Like lists, dicts are mutable, dynamic, can be nested
- Unlike lists, order doesn’t matter. Lists elements are accessed using indices; dictionary elements are accessed using keys

Dictionaries



```python
# The general syntax for a dictionary looks like this
d = {
    <key>: <value>,
    <key>: <value>,
      .
      .
      .
    <key>: <value>
}
# this cell won't run, it's just an example.
```



```python
# An example dictionary
NBA_teams = {
    "Oklahoma City": "Oklahoma City Thunder",
    "New York": "New York Knicks",
    "Brooklyn": "Brooklyn Nets",
    "Salt Lake City": "Utah Jazz"
}
```

Activity 3: Sets

Activity 4: Dictionaries

Working with real data

- Python can import, read, manipulate, and write files using built-in functions

  open("filename", "mode") opens a file with name "filename" using mode = "mode":

  - 'r': read only
  - 'w': overwrite and write
  - 'a': append
  - 'r+': both read and write

Paths and Directories

- Computers can only find a file if you tell them to look in the right folder (aka directory)
- The address of a directory on your computer is called a path
- Forward slashes (/) denote folders inside folders:
  - /Desktop/ClassHW/hw_2.ipynb: ClassHW is a directory inside the directory Desktop and contains the file hw_2.ipynb

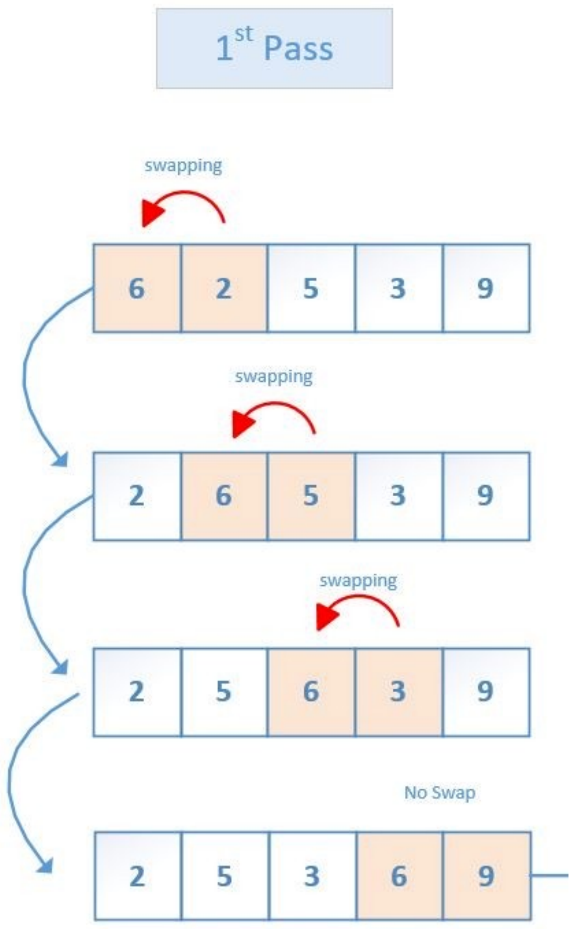
Relative vs Absolute Paths

- *Absolute* paths refer to location in file system relative to the root directory:
  - /Desktop/ClassHW/HW2/hw_2.ipynb

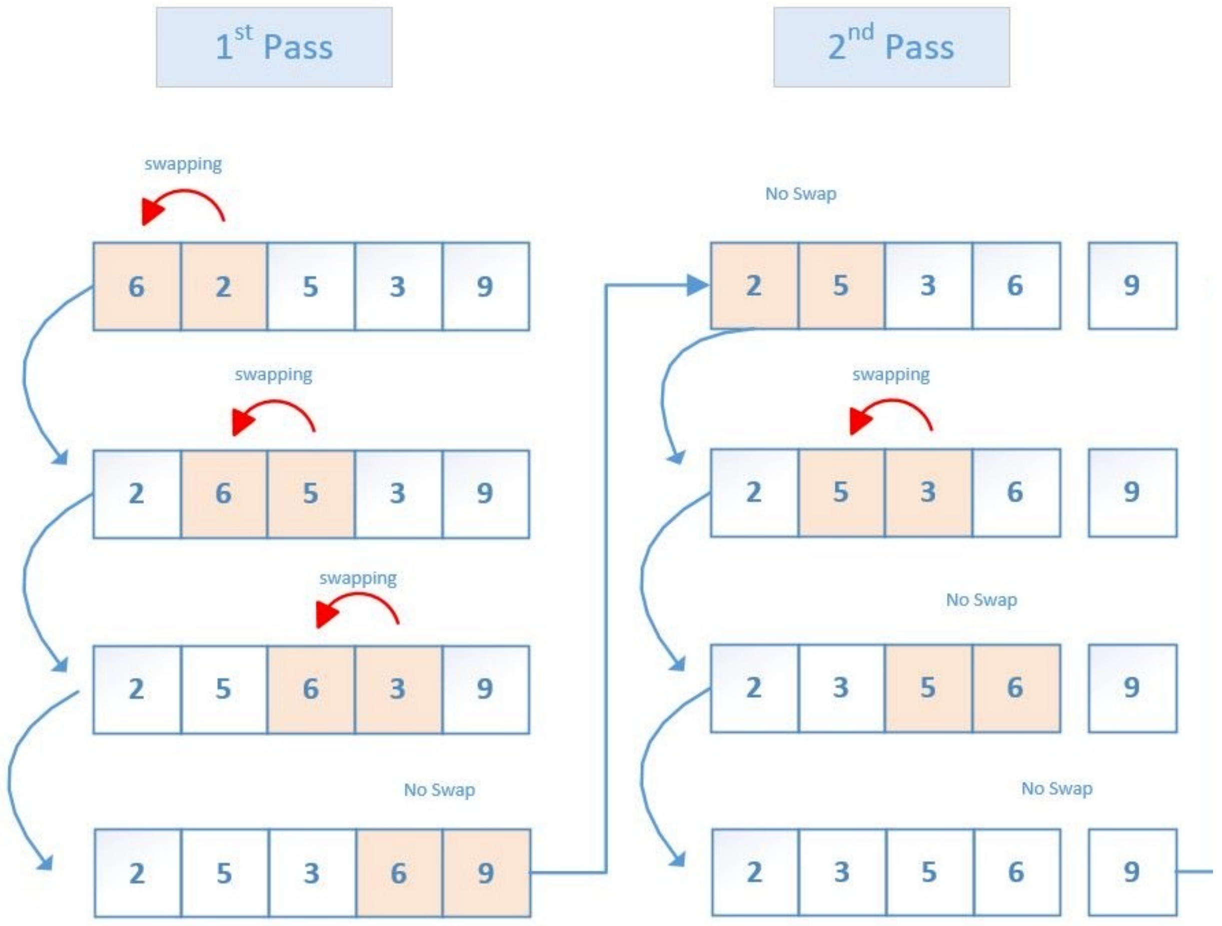
- *Relative* paths refer location in file system relative to the current directory you are working in:
  - if I'm in the ClassHW directory, then: HW2/hw_2.ipynb is the relative path of this notebook

Activity: Reading / Writing to Files

Bubble Sort


1st Pass
swapping
6 2 5 3 9
swapping
2 6 5 3 9
swapping
2 5 6 3 9
No Swap
2 5 3 6 9

Bubble Sort



1st Pass
2nd Pass
swapping
6 2 5 3 9
swapping
2 6 5 3 9
swapping
2 5 6 3 9
No Swap
2 5 3 6 9
No Swap
2 5 3 6 9
swapping
2 5 3 6 9
No Swap
2 3 5 6 9
No Swap
2 3 5 6 9

Bubble Sort

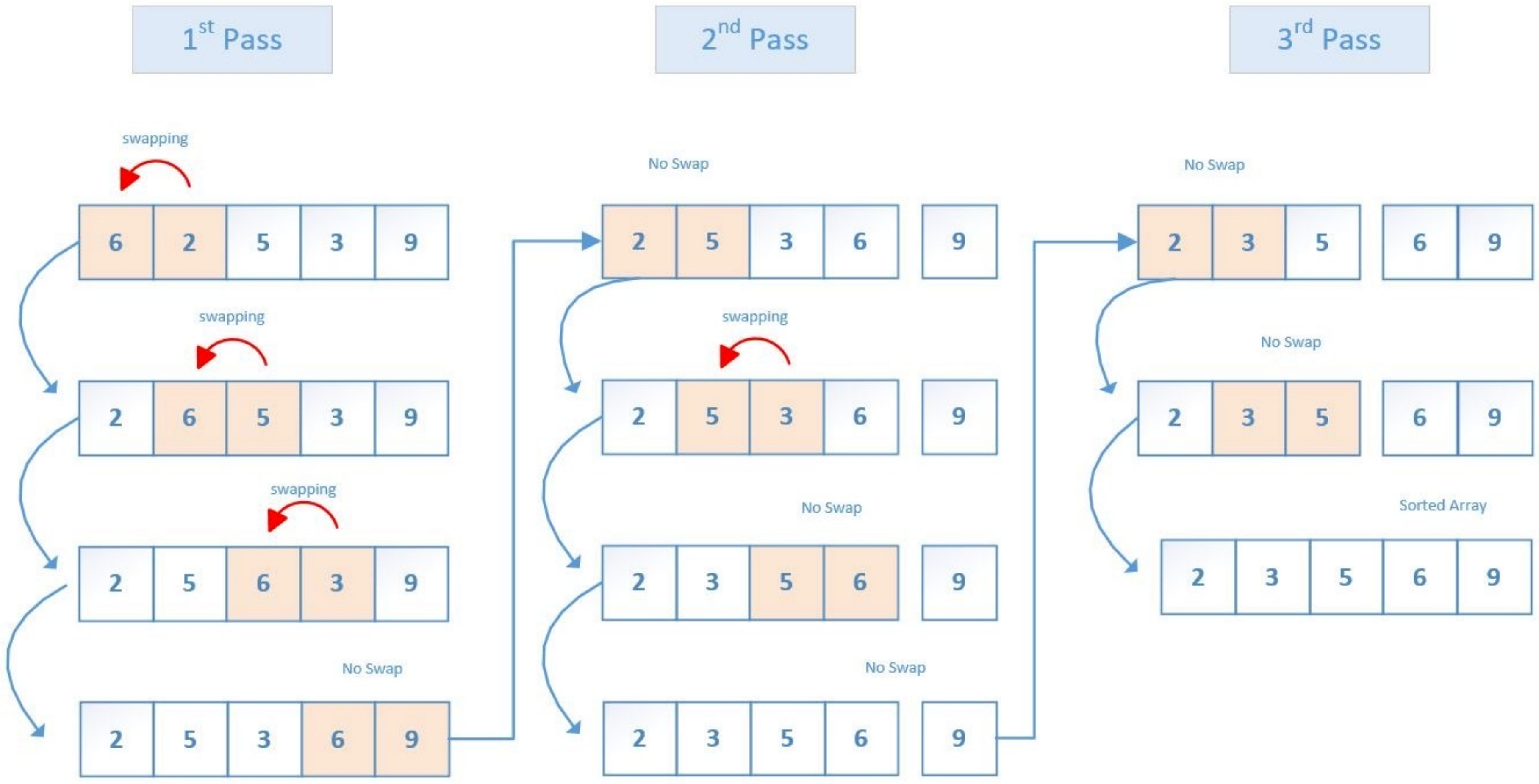
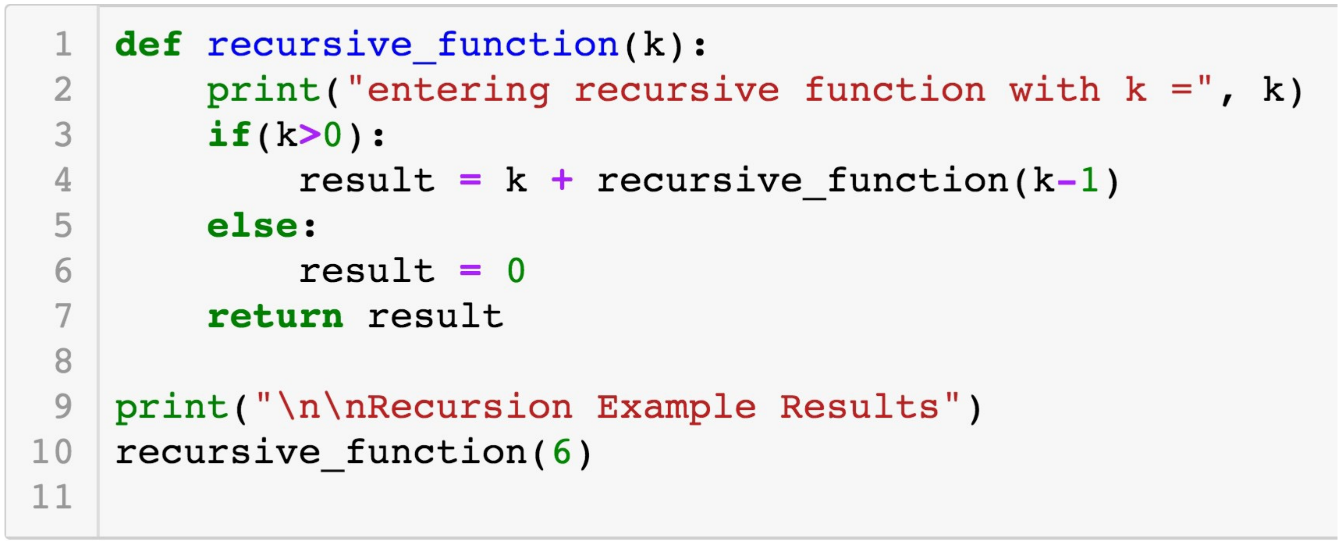


Image source: techdemic.com/bubble-sort

Activity: Bubble Sort

Recursive functions:



```python
def recursive_function(k):
    print("entering recursive function with k =", k)
    if(k>0):
        result = k + recursive_function(k-1)
    else:
        result = 0
    return result

print("\n\nRecursion Example Results")
recursive_function(6)
```

```
Recursion Example Results
entering recursive function with k = 6
entering recursive function with k = 5
entering recursive function with k = 4
entering recursive function with k = 3
entering recursive function with k = 2
entering recursive function with k = 1
entering recursive function with k = 0

21
```

Activity:

Write a recursive function that takes as input a positive integer N and calculates N!

Reflection

- How are you? How are things going?
- Are you being challenged enough? Are you seeking out enough support? If not, how can you be more proactive about getting the most out of this class?