ICS Spring 2017

Lab Session 6

Agenda

- **Warm-up**
  - **Recursion**
  - **OOP**
- **Algorithm Workshop Follow-up**

Github (yet again)

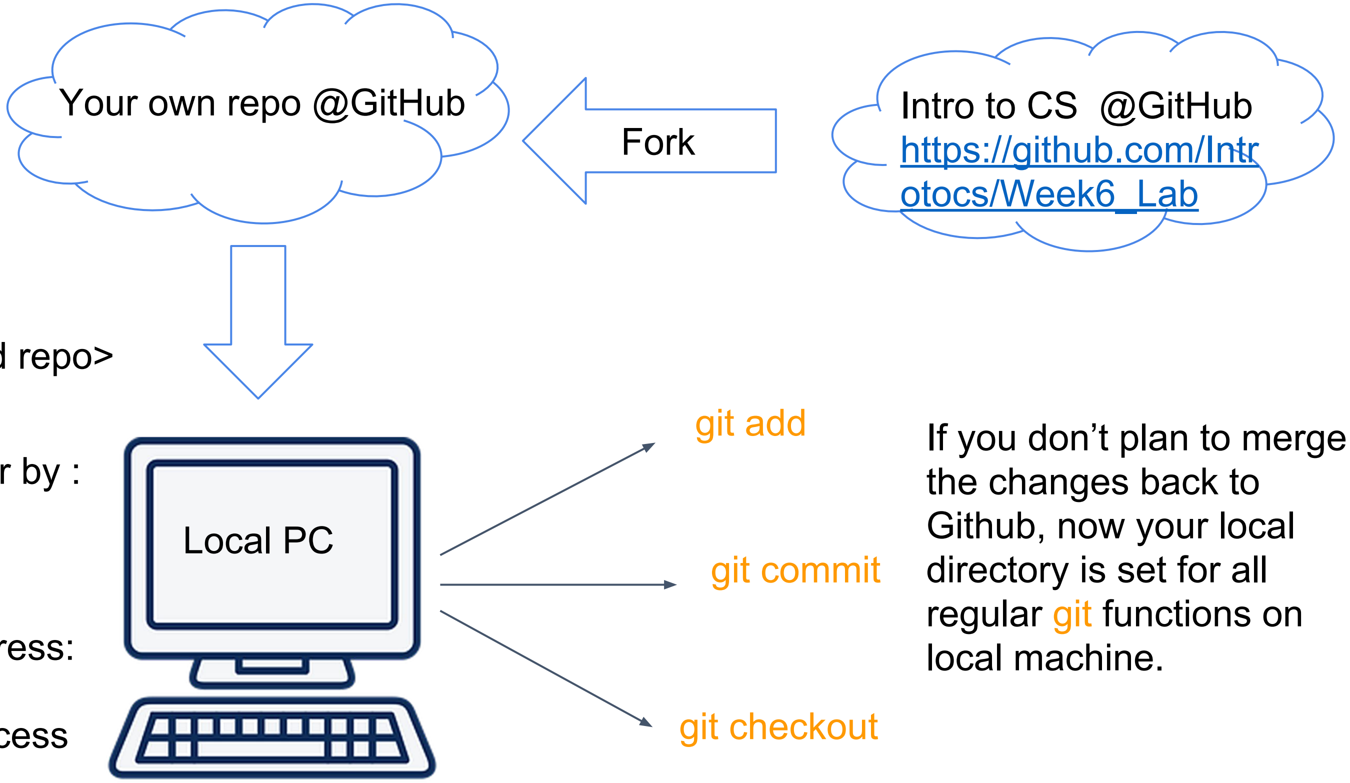
Prep:

- Fork from class (public) repo
- **git clone** to local folder

Loop:

- **git branch** “my_beautiful_code”
- Work on code; **git add/commit**
- **git merge**
- **git push** (your beautiful code is now public and everywhere!)

Downloading from GitHub



git clone <URL of your forked repo>

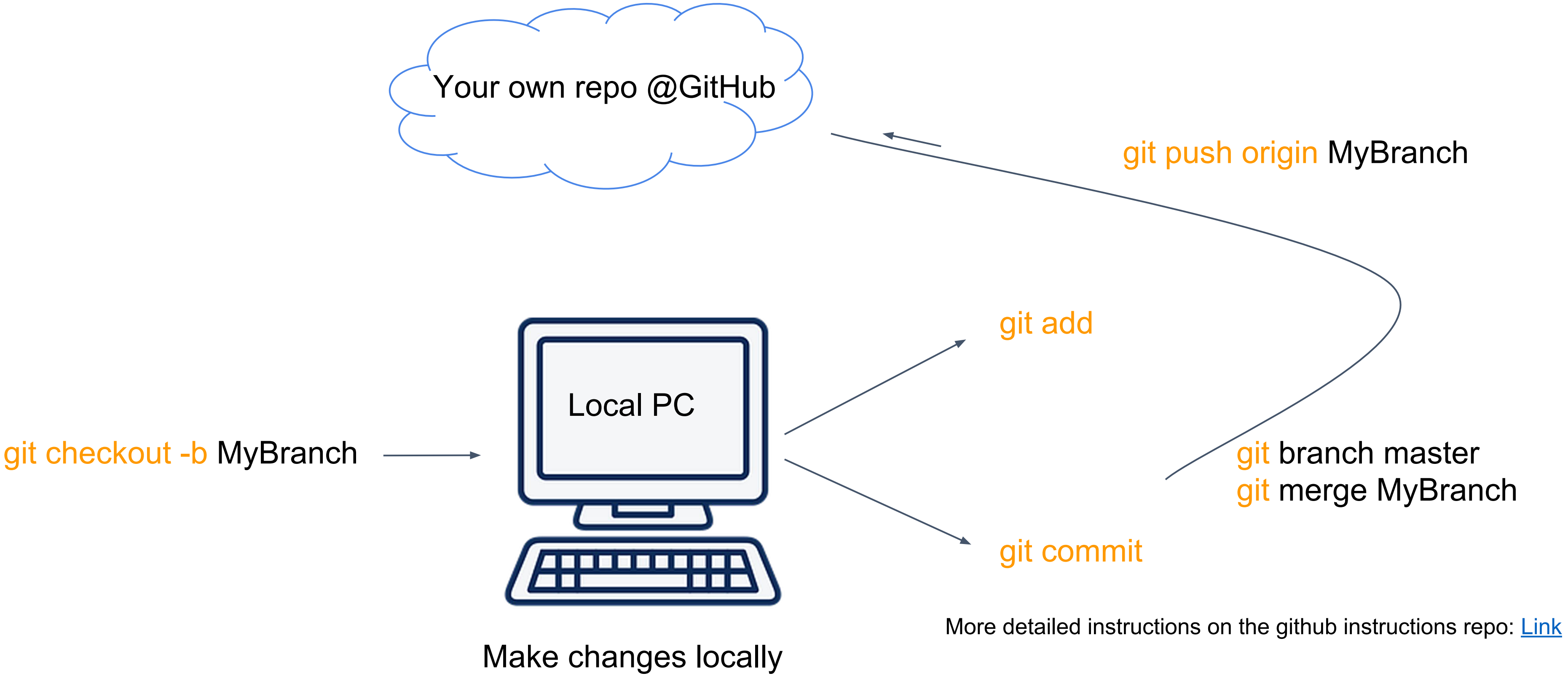
Move to the week 6 lab folder by :
cd Week6_Lab
cd → Change directory

Check the current folder address:
pwd
Navigate to it manually to access the files.

If you don't plan to merge the changes back to Github, now your local directory is set for all regular git functions on local machine.

More detailed instructions on the github instructions repo: Link

Making changes to files on your local machine and syncing with your repo on Github



More detailed instructions on the github instructions repo: Link

Recursion Exercise

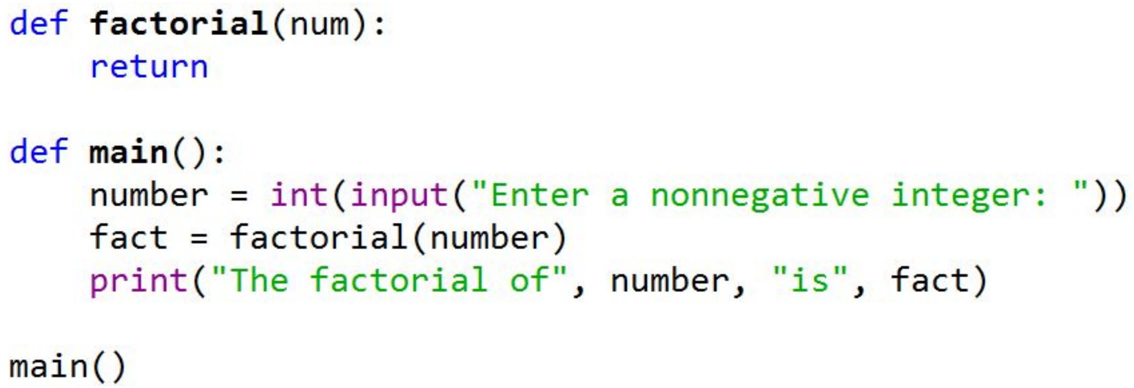
In-class Exercises Q1: Factorial (fact.py)

Write a program with recursion to find the factorial of provided non-negative integer n.

Example:

Given n = 5

Return 120



```python
def factorial(num):
    return

def main():
    number = int(input("Enter a nonnegative integer: "))
    fact = factorial(number)
    print("The factorial of", number, "is", fact)

main()
```

OOP Warm-up Exercise

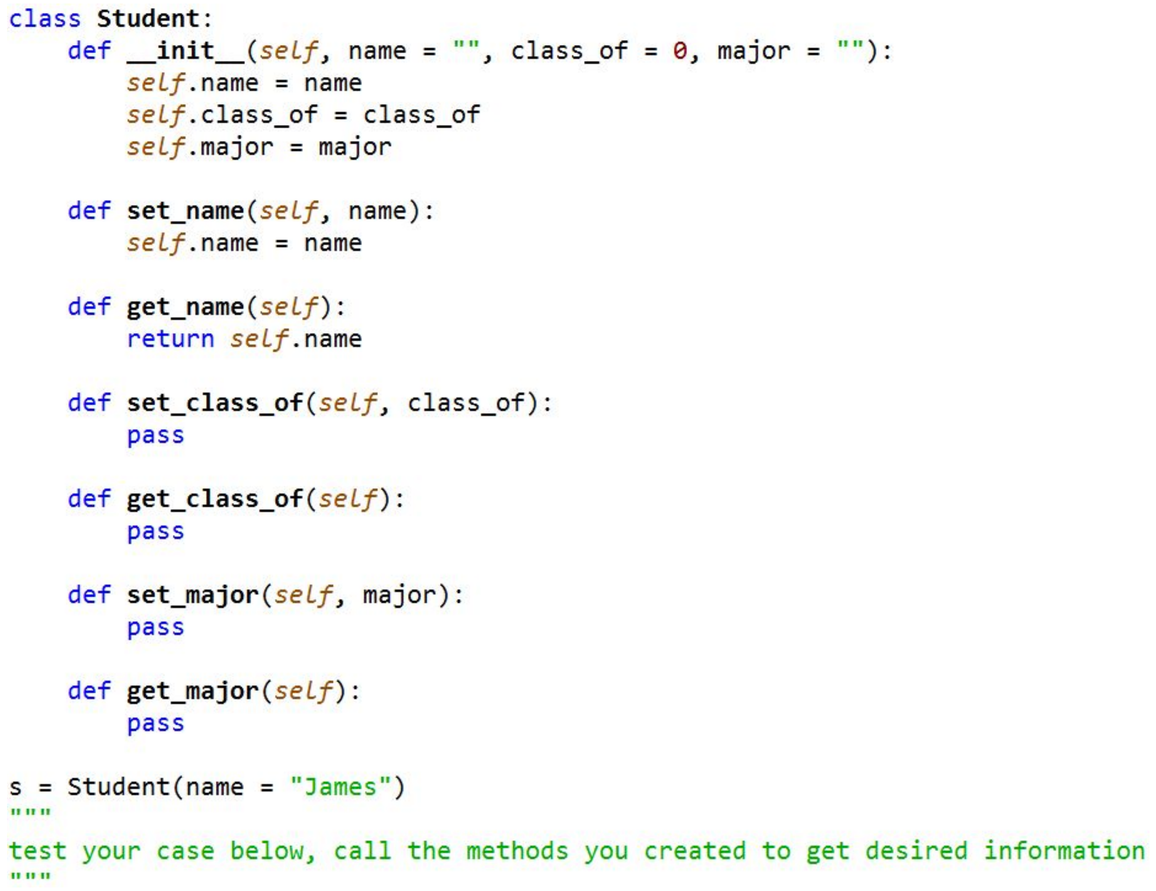
For OOP tutorial videos, click here

In-class Exercises Q2: Student Information

(starting code in next slide)

OOP Warm-up Exercise

- Starting Code
  - Write your methods
  - Make sure you understand
    - __init__
    - Getters
    - Setters



```
class Student:
    def __init__(self, name = "", class_of = 0, major = ""):
        self.name = name
        self.class_of = class_of
        self.major = major

    def set_name(self, name):
        self.name = name

    def get_name(self):
        return self.name

    def set_class_of(self, class_of):
        pass

    def get_class_of(self):
        pass

    def set_major(self, major):
        pass

    def get_major(self):
        pass

s = Student(name = "James")
"""
test your case below, call the methods you created to get desired information
"""
```

Algorithm Workshop Follow-up

Problem 1. Number Placement

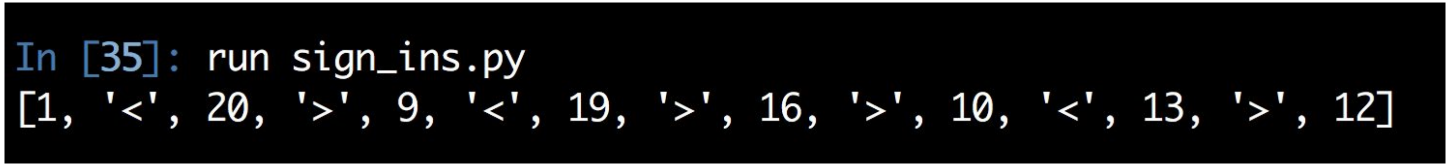
1. Number Placement

- *n* numbers; *n* - 1 preset inequality sign
- **Goal**: insert the numbers so that the inequality hold

Example:

Numbers: [2, 3, 0, 1, 5]; Signs: ['<', '>', '<', '<']

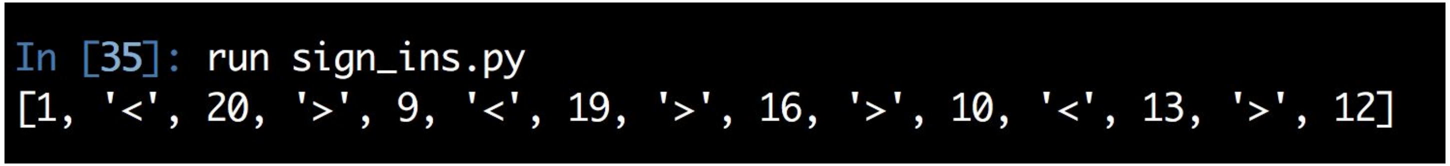
Solution: 0 **<** 5 **>** 1 **<** 2 **<** 3.



```
In [35]: run sign_ins.py
[1, '<', 20, '>', 9, '<', 19, '>', 16, '>', 10, '<', 13, '>', 12]
```

Hints

- sort the numbers into a list
- go from the left to right:
    - if the sign is “<”
        - pop the smallest to the left of “<”
    - if the sign is “>”
        - pop the largest to the left of “>”
    - pop() ***removes*** the number from the list

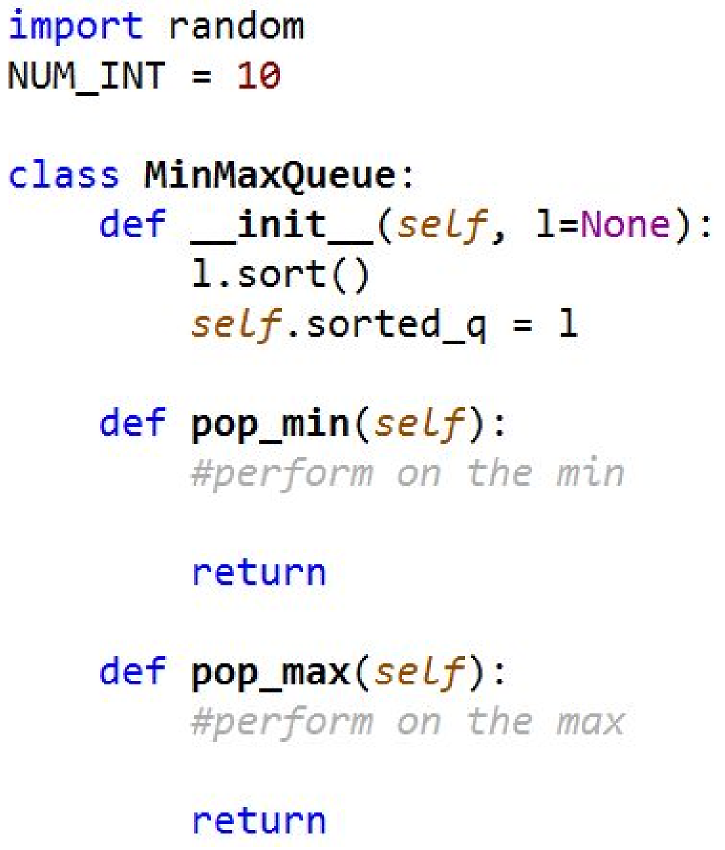


```
In [35]: run sign_ins.py
[1, '<', 20, '>', 9, '<', 19, '>', 16, '>', 10, '<', 13, '>', 12]
```

Starting Code

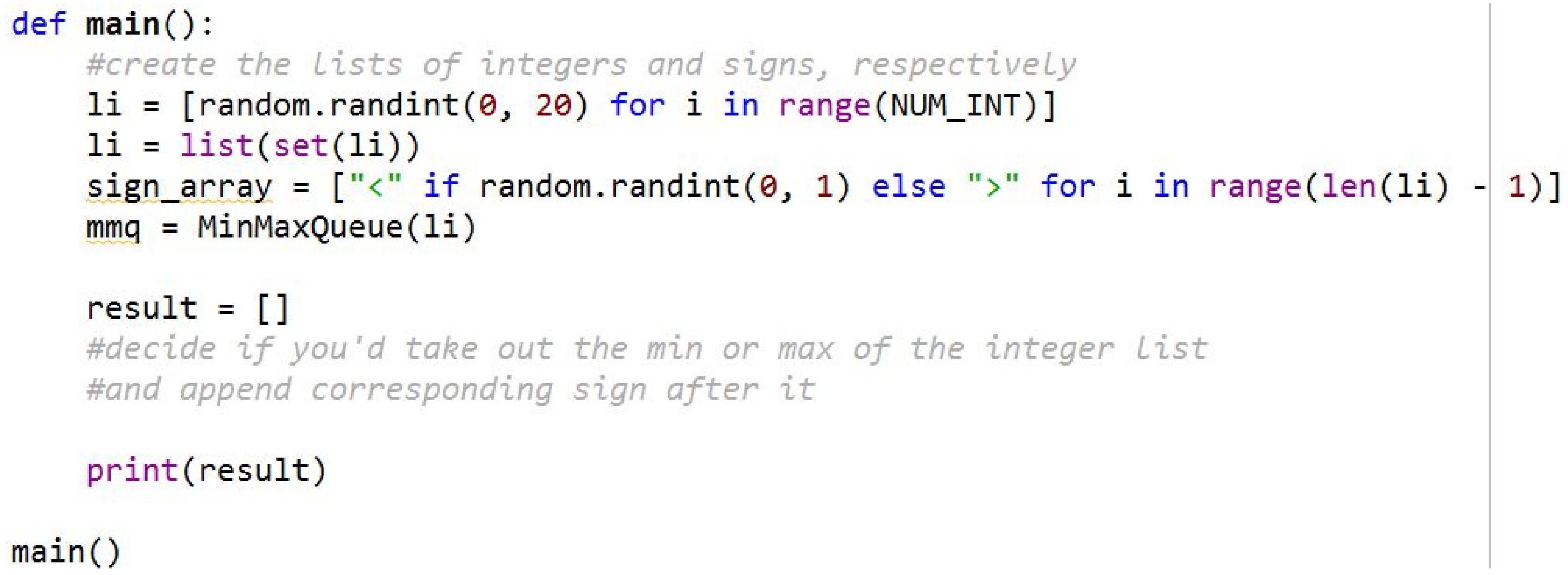
- Build your Class
- Understand which argument(s) you will pass
- Define your methods
- See the hints from comments if confused

Once these are done, play it out in console



```python
import random
NUM_INT = 10

class MinMaxQueue:
    def __init__(self, l=None):
        l.sort()
        self.sorted_q = l

    def pop_min(self):
        #perform on the min

        return

    def pop_max(self):
        #perform on the max

        return
```

Starting Code



```python
def main():
    #create the lists of integers and signs, respectively
    li = [random.randint(0, 20) for i in range(NUM_INT)]
    li = list(set(li))
    sign_array = ["<" if random.randint(0, 1) else ">" for i in range(len(li) - 1)]
    mmq = MinMaxQueue(li)

    result = []
    #decide if you'd take out the min or max of the integer list
    #and append corresponding sign after it

    print(result)

main()
```

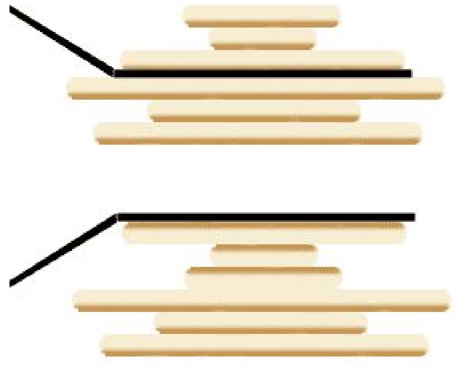
- Understand how the lists of integers and signs are generated
- Goal: create the object, manipulate it with its methods

Algorithm Workshop Follow-up

Problem 2. Pancake Sorting

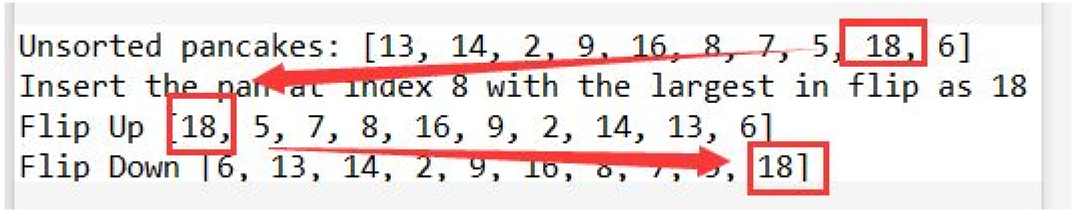
2. Pancake Sorting

- *n* pancakes of different sizes, randomly stacked
- Allowed action: slip a spatula under one pancake, and flip
- **Goal**: sort the pancakes (smallest at the top)



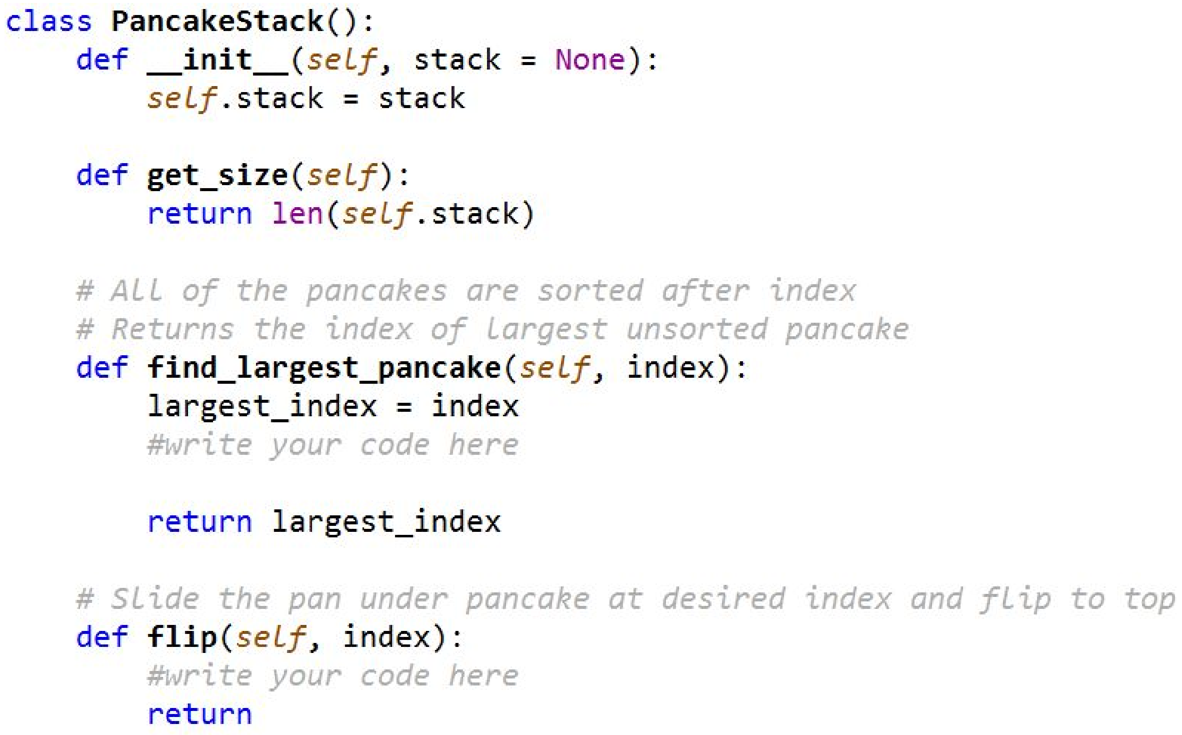
Hints

- move down level by level
  - Find the largest *out of order* pancake
  - Flip it to the bottom
    - you may need to flip it to the top of the stack first
  - Repeat step one until the stack is ordered.



```
Unsorted pancakes: [13, 14, 2, 9, 16, 8, 7, 5, 18, 6]
Insert the pan at index 8 with the largest in flip as 18
Flip Up [18, 5, 7, 8, 16, 9, 2, 14, 13, 6]
Flip Down [6, 13, 14, 2, 9, 16, 8, 7, 5, 18]
```

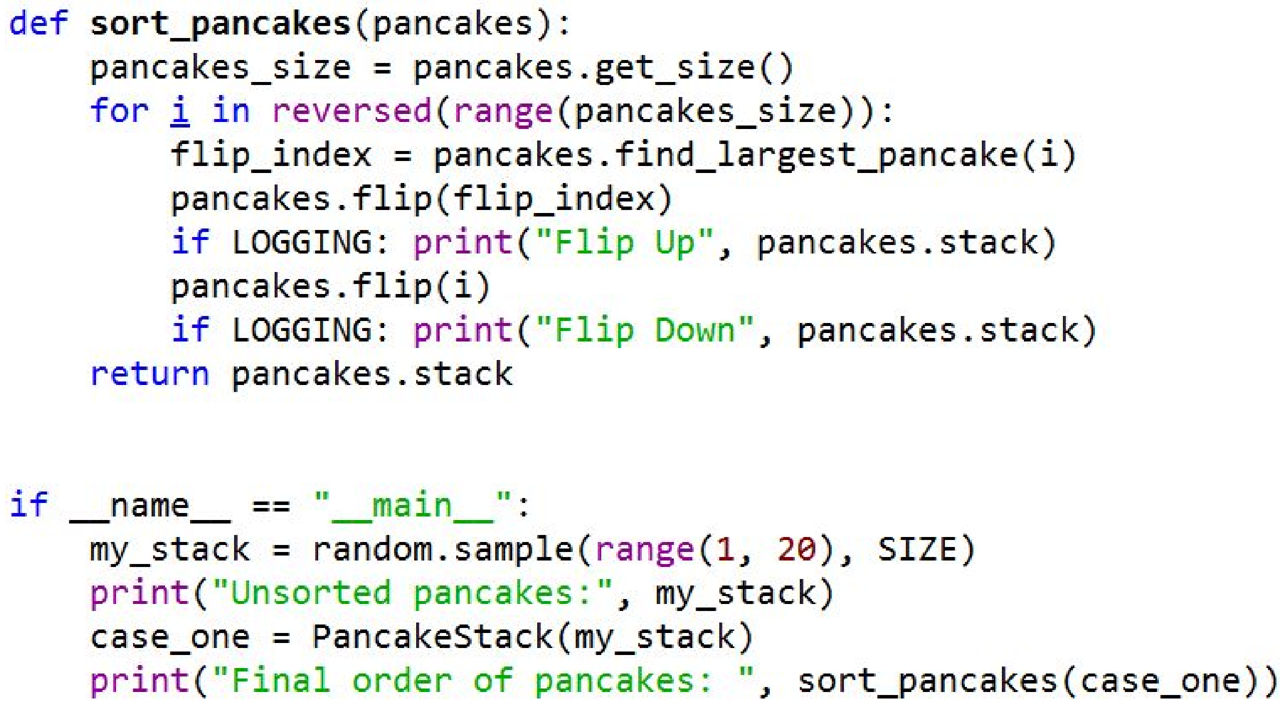
Starting Code



```python
class PancakeStack():
    def __init__(self, stack = None):
        self.stack = stack

    def get_size(self):
        return len(self.stack)

    # All of the pancakes are sorted after index
    # Returns the index of largest unsorted pancake
    def find_largest_pancake(self, index):
        largest_index = index
        #write your code here

        return largest_index

    # Slide the pan under pancake at desired index and flip to top
    def flip(self, index):
        #write your code here
        return
```

- Build your Class
- Understand which argument(s) you will pass
- Define your methods
- See the hints from comments if confused

Starting Code



```python
def sort_pancakes(pancakes):
    pancakes_size = pancakes.get_size()
    for i in reversed(range(pancakes_size)):
        flip_index = pancakes.find_largest_pancake(i)
        pancakes.flip(flip_index)
        if LOGGING: print("Flip Up", pancakes.stack)
        pancakes.flip(i)
        if LOGGING: print("Flip Down", pancakes.stack)
    return pancakes.stack


if __name__ == "__main__":
    my_stack = random.sample(range(1, 20), SIZE)
    print("Unsorted pancakes:", my_stack)
    case_one = PancakeStack(my_stack)
    print("Final order of pancakes: ", sort_pancakes(case_one))
```

- We randomly generated the pancake list for you
- There’s also the sorting function
    - Which takes care of your procedures created in your Class

2nd Approach on Pancake Sorting:

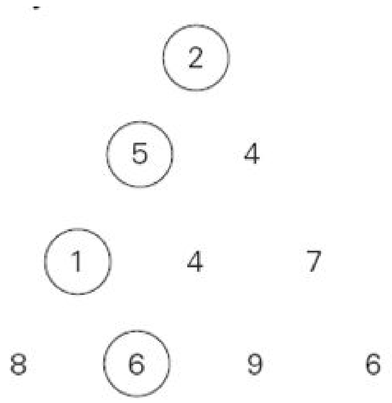
- move down level by level
  - if the current level has smaller size:
    - flip above
    - and then flip this pancake up to the top
    - then flip to its proper location
- Implementation of this approach will be bonus in Problem Set 6

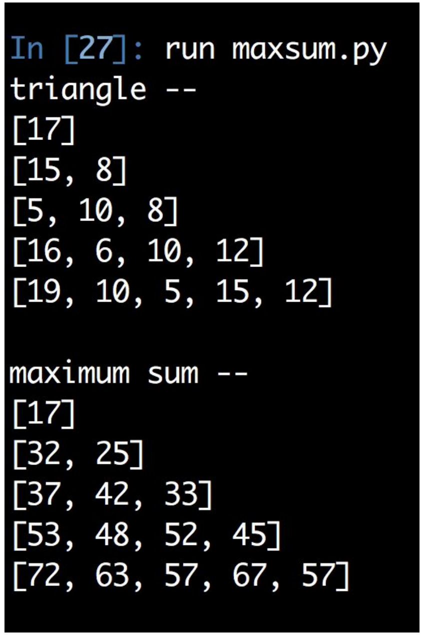
Algorithm Workshop Follow-up

Problem 3. Maximum Sum Descent

3. Maximum Sum Descent

- Positive integers in a triangle
- **Goal**: a descent from the root to the base, with the largest sum.





```
In [27]: run maxsum.py
triangle --
[17]
[15, 8]
[5, 10, 8]
[16, 6, 10, 12]
[19, 10, 5, 15, 12]

maximum sum --
[17]
[32, 25]
[37, 42, 33]
[53, 48, 52, 45]
[72, 63, 57, 67, 57]
```

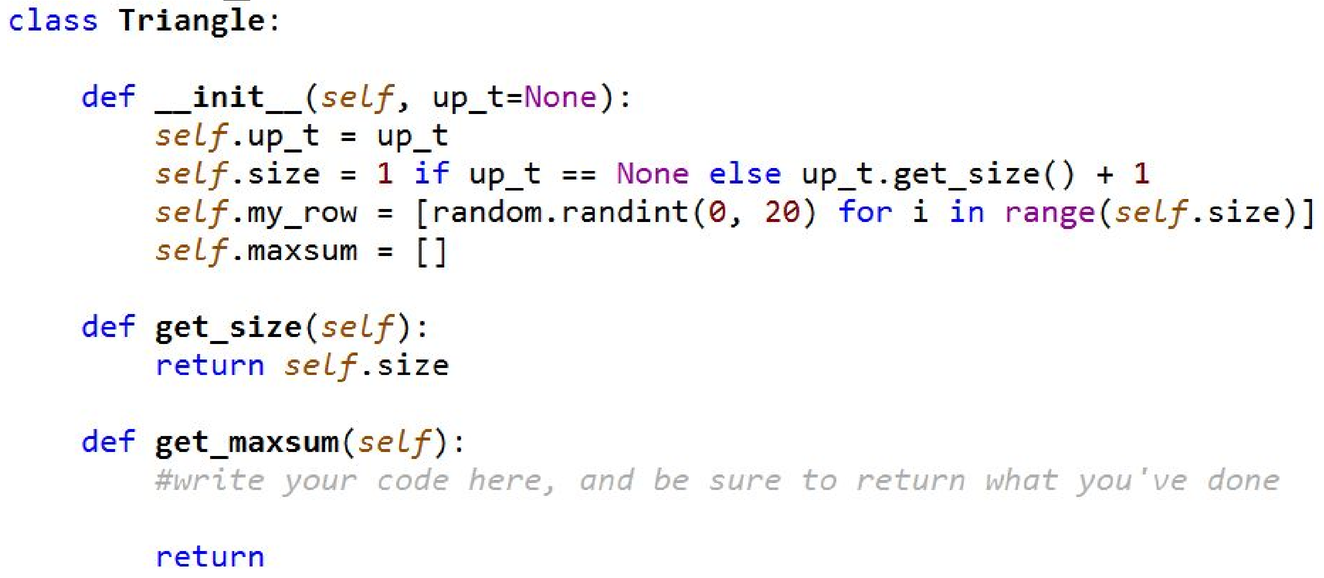
Hints: get_maxsum()

E.g.: get_maxsum(7); (7 is the last row)

- Scan numbers in row 7 (call it self.my_row)
- Suppose you have maxsum of the triangle above
  - By calling self.up_triangle.get_maxsum() (**recursion!!**)
  - For each x in self.my_row, max the distances of its parent, that's maxsum to x from root

This is OOP + recursion

Starting Code



```python
class Triangle:

    def __init__(self, up_t=None):
        self.up_t = up_t
        self.size = 1 if up_t == None else up_t.get_size() + 1
        self.my_row = [random.randint(0, 20) for i in range(self.size)]
        self.maxsum = []

    def get_size(self):
        return self.size

    def get_maxsum(self):
        #write your code here, and be sure to return what you've done

        return
```

- Build your Class
- Understand which argument(s) you will pass
- Create your methods
    - Here the core one is `get_maxsum()`
- See the hints from comments if confused

Starting Code

- We’ve generated the triangle for you
  - How was this triangle object created?
  - Make sure you understand what __init__ does
  - maxsum() is performed line by line
    - From the bottom n to the top

```python
def print_triangles(ts):
    for t in ts:
        print(t.my_row)

def print_maxsum(ts):
    for t in ts:
        print(t.maxsum)

def main():
    tri = []
    for i in range(TRI_DEPTH):
        try:
            tri.append(Triangle(tri[i-1]))
        except:
            tri.append(Triangle())

    print("triangle -- ")
    print_triangles(tri)

    tri[TRI_DEPTH - 1].get_maxsum()
    print("\nmaximum sum -- ");
    print_maxsum(tri)

main()
```