Problem Solving Session

- The remainder of today's class will comprise the ***problem solving session*** (***PSS***).
- Your instructor will divide you into ***teams of 3 or 4 students***.
- Each team will ***work together*** to solve the following problems over the course of ***20-30 minutes***.
  - You may work on paper, a white board, or digitally as determined by your instructor.
  - You will submit your solution by pushing it to GitHub before the end of class.
- Your instructor will go over the solution before the end of class.
- If there is any time remaining, you will begin work on your homework assignment.

Class participation is a significant part of your grade (20%). This includes in class activities and the problem solving session.

Your Course Assistants will grade your participation by verifying that you pushed your solutions before the end of the class period each day.

Problem Solving 1

Write a *recursive* function that at least declares parameters for `a_list` and an `amount` (you may add additional parameters).

The function should add the given `amount` to each value in `a_list`.

```
>>> print(a_list)
[3, 4, 2, 5, 1]
>>> add_to_all(a_list, 5)
>>> print(a_list)
[8, 9, 7, 10, 6]
>>> _
```

```
def add_to_all(a_list, amount, i=0):
    if i >= len(a_list):
        return None
    a_list[i] += amount
    add_to_all(a_list, amount, i + 1)
```

Problem Solving 2

Rewrite the function below so that it uses *recursion* rather than a loop. You may add additional parameters if needed.

```
def reverse(a_string):
  reversed = ""
  for i in range(len(a_string)):
    reversed = a_string[i] + reversed

return reversed
```

```
def reverse_recursion(a_string, i = 0):
     if i >= len(a_string):
          return “”

     reversed = reverse_recursion(a_string, i+1) + a_string[i]

     return reversed
```

Problem Solving 3

The Fibonacci Sequence is described by the recursive mathematical formula shown below. Examine the formula and then answer the questions to the right.

F(N) WHERE N<=0 IS UNDEFINED

F(1) = 0

F(2) = 1

F(N) = F(N-1) + F(N-2)

Assume you are writing a naive implementation. Write the function signature (name & parameters).

```
def fibonacci_function(n):
```

Which part(s) of the formula would be the ***base case(s)*** in your implementation?

THE BASE CASE OF THIS FORMULA WOULD BE WHERE N <= 0 (UNDEFINED).

2ND BASE CASE: N==1 RETURNS 0

3RD BASE CASE: N==2 RETURNS 1

Which part(s) of the formula would be the ***recursive case(s)*** in your implementation?

Everything else would be the recursive case of the formula
(n is other returns recursive call with n-1 plus recursive call with n-2)

```
def fibonacci_function(n):
     if n <= 0:
          return None
     elif n == 1:
          return 0
     elif n == 2:
          return 1
     else:
          return fibonacci_function(n-1) +
fibonacci_function(n-2)

def main():
     print(fibonacci_function(10))

if __name__ == “__main__”:
     main()
```

Problem Solving 4

The Fibonacci Sequence is described by the recursive mathematical formula shown below. Work together with your team to write a function that provides a naive implementation of the formula.

F(N) WHERE N<=0 IS UNDEFINED

$F(1) = 0$

$F(2) = 1$

$F(N) = F(N-1) + F(N-2)$

Problem Solving 5

Work together with your team to write a pytest unit test that fully tests your naive implementation of the Fibonacci sequence. You should include at least one test function for your base case(s) and the recursive case(s).

If you are working digitally and need more space, duplicate this slide.

F(N) WHERE N<=0 IS UNDEFINED

F(1) = 0

F(2) = 1

F(N) = F(N-1) + F(N-2)

Begin by working out the first 10 numbers in the sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

```
import fibonacci
def test_fib_ten():
        #setup
        expected = 34

        #invoke
        actual = fibonacci.fibonacci_function(10)

        #analyze
        assert expected == actual

def test_base_invalid():
        #setup
        expected = None

        #invoke
        actual = fibonacci.fibonacci_function(-69)

        #analyze
        assert expected == actual
```