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

Problem Solving Team Members

If you are working digitally, record the name of each of your problem solving team members here.

Do not forget to ***add every team member's name***! Your instructor (or course assistant) may or may not use this to determine whether or not you participated in the problem solving session.

Bryan Navarro
Jackson Palmer
Ryan Robison

Problem 1

Consider a simple number guessing game during which the player is given 3 chances to guess a number between 1 and 10. If any guess that the player makes is correct, the game is over.

If they fail to guess the correct answer within 3 guesses, the answer is printed and the game is over.

You will implement a function that validates *one* of the player's guesses and ***returns*** a *string message* indicating whether or not it is out of range (less than 1 or greater than 10), too low, too high, or correct.

Assuming that you are using incremental development to write the function, what increments would you need to write to completely implement the entire game? List as many as you can think of.

Check with your instructor or a Course Assistant before moving to the next step.

1. use random to generate an int between 1-10. User must then input an int. If the users int == random int, print a string saying “you’re correct” If users int > or < random int, have a new variable called checkGuess++. Once this int variable reaches a score of 3, send a print statement saying “game over.” If user input int is higher than 10 or lower than 1,
   ex (0, 12, -10)
   then print “out of range”
   if users int is greater than the randomly generated int, print “too high”, Similarly, if the users int is less than the random int, print “too low”

```
def check_guess(guess,answer):
   if guess > 10 or guess < 1:
      return(“out of range”)
   if guess < answer:
      return(“too low”)
   if guess > answer:
      return(“too high”)
   if guess == answer:
      return(“correct”)
```

Problem 2

Begin implementing a function named "`check_guess`" that declares parameters for the `answer` and the player's `guess`.

Choose ***one*** of the increments that you and your team described in the previous problem and write ***only*** the minimum code necessary to implement that increment.

Next, write a pytest test function that verifies that your function is working properly. Remember, a good test has three parts:

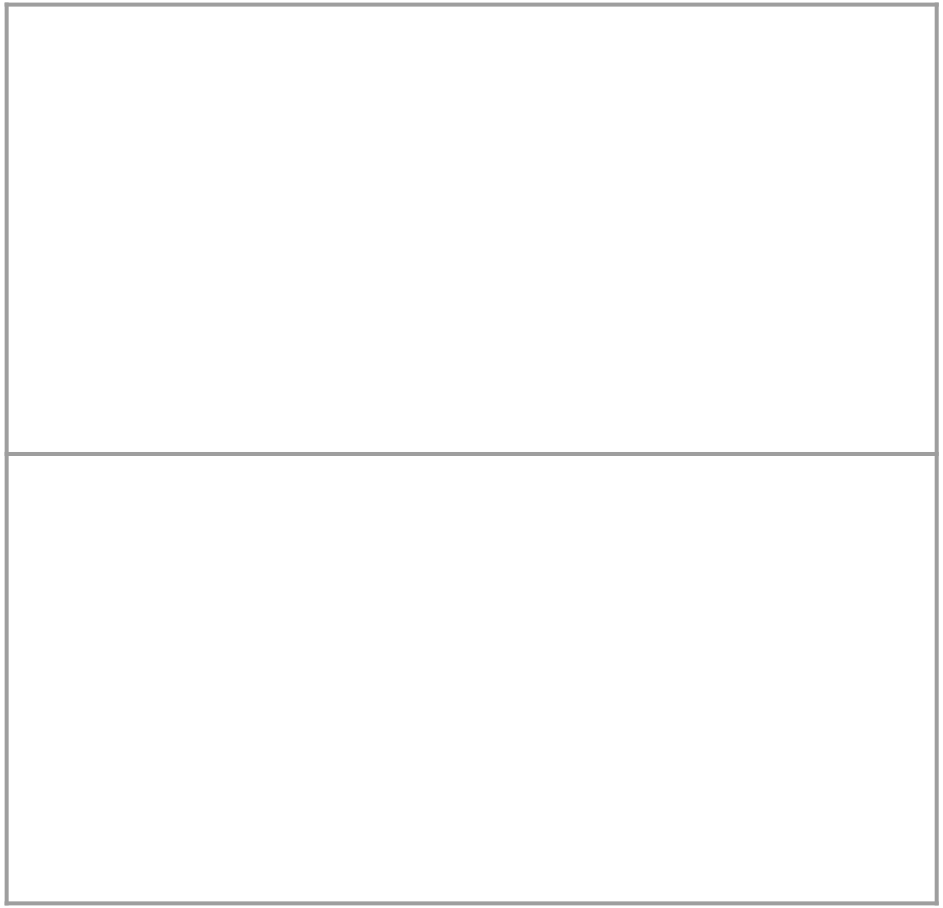
- *setup* - create variables needed for input and to verify the correct results.
- *invoke* - call the function under test.
- *analyze* - verify that the function produced the correct results.

Problem 3

Repeat the steps of the previous problem with a new increment and unit test.

How will the function or the new unit test need to change compared to the first test to verify that the function returns the correct value?

Hint: if either the test or the function do not need to change at all, either your previous test was testing more than one thing, or you don't need this new test.



```
def rand():
  X = random.randint(1, 100)
```

It should be about 7 minimum guesses because each guess you guess in the middle of the numbers you have left then you are told higher or lower and you keep splitting the amount left for example 50, lower ,25, lower, 12, lower 6, higher 9, lower 8, 7, this way u keep splitting the pool of numbers in half.

Problem 4

Python's `random` module provides several functions that generate pseudorandom numbers within a given range, including:

- `random()` - returns a random floating point value in the range [0.0, 1.0]
- `randrange(a, b)` - returns a random integer value in the range [a, b).
- `randint(a, b)` - returns a random integer value in the range [a, b]

Using the random module, write the code necessary to generate a pseudorandom integer between 1 and 100 (inclusive).

What is the minimum number of guesses that a person would need to guarantee that they could guess the number?

Problem 5

Once your guess validating function is complete, you will be able to implement a `main` function that allows a player to play your game. Working together with your team, sketch out an algorithm using ***pseudocode*** to implement the guessing game. The player gets up to three guesses to guess the number, but the game ends if they guess correctly.

You should begin by generating a random number between 1 and 10 for the player to guess.

You should print feedback to the user after every guess.

```
set answer to random.randint(1,10)
ask for guess input
checker equals return of check_guess
guesses = 1
print(checker)
if checker != “correct” and guesses < 3
ask for guess input
checker equals return of chec_guess
guesses = guesses + 1
print(checker)
if checker != “correct” and guesses < 3
ask for guess input
checker equals return of chec_guess
guesses = guesses + 1
print checker
if checker == “correct”
print(checker)
exit
if guesses >= 3
print (“you lose correct answer was ”, answer)
exit
```