Follow the instructions below:

- Write your solution to each question in the cell just below the question.
- If you need any space to try code that will not be graded, use the space at the bottom below the word “Scratchwork”.
- There are multiple ways to solve many of these problems, so we will give credit (including partial credit) based on how well your code solves the problem given.
- You should also use good programming style, such as descriptive variable/function names and axes labels.
- Comments are only required if specifically requested, but feel free to use them whenever you want.
- If you’re not sure how to code something, use comments to describe what the code should do and code as much as you can to earn partial credit.

By writing your name below, you acknowledge that you read the instructions above and have not looked at anyone else’s code or any other resources outside this worksheet.

# Name: Kristin McCully
# Section: none (I'm the LS30 coordinator!)
# Date: 11/22/2017

Practice Exam # 1

1. (10 points) Write a function that prints “It’s hot” if the temperature is greater than 85, “It’s cold” if temperature is less than 65, and “Not bad” if the temperature is between 65 and 85, inclusive. Test the function with three different temperatures.

```python
# 1

def weather (temp):
    if temp > 85: # if high temp
        print "It's hot"
```
elif temp < 65:  # if low temp
    print "It's cold"
else:  # if between 65 and 85 inclusive
    print "Not bad"

weather (85)  # testing
weather (43)
weather (65)
Not bad
It's cold
Not bad

2. (10 points) You are studying populations of penguins and marine iguanas on a beach in the Galapagos. Over five years, the penguin population at your study site has been 62, 93, 75, 56 and 76. In the same years, the marine iguana population has been 34, 21, 15, 25 and 34. Plot the system's states in penguin-marine iguana space, making the points green and large.

penguins = [62, 93, 75, 56, 76]  # define list for penguin population
iguanas = [34, 21, 15, 25, 34]  # define list for iguana population
list_plot(zip(penguins, iguanas), axes_labels = ["Number of Penguins", "Number of Iguanas"], size = 40, color = "green")

3. (10 points) The simulation script below has five errors. Correct the errors and explain what each line does in a comment.

```python
var("N,P")
t = srange(0,100,0.1)
sol=desolve_odeint([0.5*N - 0.01*N*P, 0.5*0.01*N*P - 0.2*P],
ics=[50,75], dvars=[N,P], times=t1)
list_plot((t,sol[:,0]) + plot(zip(t,sol[:,1]), color="red")
```

# 3.
var("N,P") # Define N and P as symbolic variables
t = srange(0,100,0.1) # Create list of time steps from 0 to 100 in steps
sol=desolve_odeint([0.5*N - 0.01*N*P, 0.5*0.01*N*P - 0.2*P], ics=[50,75], dvars=[N,P], times=t) # Simulate differential equation for given initial conditions, time points, variables
list_plot(zip(t, sol[:,0])) + list_plot(zip(t, sol[:,1]), color="red") # Plot time series (could add axes labels, but not actually required when just correcting errors)

# Errors:
# Parentheses in line 2
# t1 instead of t inside desolve_odeint in line 3
# Missing * in 0.2P in desolve_odeint in line 3
# No zip in first list_plot in line 4
# Plot instead of list_plot in line 4

4. (10 points) Write a script that calculates the factorial of an integer \( n \) (written \( n! \)). To calculate a factorial, we take the number, \( n \), and multiply it by all of the integers between 1 and \( n \), inclusive. For example, \( 2! = 2 \times 1 = 2 \) and \( 3! = 3 \times 2 \times 1 = 6 \). Test the script with two different values of \( n \).

# 4
n = 3 # Assign value to variable n
product = 1 # Create variable product that will hold product as it's calculated
for i in srange(1,n+1,1): # Run for loop through integers from 1 to n, inclusive
    product = product * i # Multiply product by counter variable i
print product # I expect that 3! = 3*2*1 = 6
Test # 2

# Test # 2

n = 5 # Assign value to variable n
product = 1 # Create variable product that will hold product as it's calculated

for i in srange(1,n+1,1): # Run for loop through integers from 1 to n, inclusive
    product = product * i # Multiply product by counter variable i

print product # I expect that 5! = 5*4*3*2*1 = 20 * 6 = 120

# Confirmation with built-in function

factorial(3) # I guessed there might be a function called factorial and used factorial? to look up its syntax. So I can check that my script works!

factorial(5)

# Note: this is not required, but nice if you figure it out!

6
120
6
120

Note: example solution for #5 is in separate document on while loops.

%hide

%html

<big><b><u>Practice Exam # 2</u></b></big><br>

Practice Exam # 2

1. (10 points) Plot the function $f(x) = x^2$ in red for values of $x$ ranging between -10 and 10. Plot the function $g(x) = x^3$ in green for values of $x$ ranging between -5 and 5. Overlay these plots. (Note: you are not required to define the functions.)

```
# 1

plot(x^2,(x,-10,10), color = "red", axes_labels = ["x","function"],
    legend_label = "f(x)") + plot(x^3, xmin = -5, xmax = 5, color = "green",
    legend_label = "g(x)")
```

# Using (x,-10,10) and xmin = -10, xmax = 10 do the same thing!
# Note: legend_label isn't required, but always good!
2. (10 points) The script below should iterate the function \( f(x) = 3x \) five times with an initial value of 1, but has five errors. Correct the errors and explain what each line does in a comment. If the script works correctly, it should generate the output at the bottom of the script. In a new cell, convert the script into a function that takes the number of iterations as input and returns the list of values. Test the function for three different numbers of iterations.

```python
mult3 = [1] nums = srange(0,5) for n in nums
    test = 3*mult3[n] append mult3(test)
Mult3 # Desired output: [1, 3, 9, 27, 81, 243]
```

Note: this example solution is for an old version that used for loops. The solution to the new version with while loops is in the separate document for while loop exercises.

```
# 2
mult3 = [1] # Create list containing just initial value 1
nums = srange(0,5) # Create list with numbers 0,1,2,3,5
for n in nums: # Run for loop that has variable n take values in \nums
    test = 3*mult3[n] # Assign 3 * last element in mult3 list to variable test
    mult3.append(test) # Add test to end of list mult3
mult3 # call mult3 to see what it contains
```

# Errors:
# Line 2: missing parentheses at end of srange()
# Line 3: no colon (:) 
# Line 4: used i instead of n 
# Line 5: list and append switched 
# Line 6: mult3 is capitalized

[1, 3, 9, 27, 81, 243]

# Purpose: iterate function \( f(x) \) a given number of times
# Input: number of iterations num_iter
# Output: result of iterating f(x) num_iter times

def iter3x (num_iter):
    mult3 = [1] # Create list containing just initial value 1
    nums = srange(0,num_iter) # Create list with numbers 0,1,2,3,5, up to num_iter
    for n in nums: # Run for loop that has variable n take values in
        test = 3*mult3[n] # Assign 3 * last element in mult3 list to
        variable test
        mult3.append(test) # Add test to end of list mult3
    return mult3 # call mult3 to see what it contains

# testing
iter3x (5)
iter3x (2)
iter3x (10)
[1, 3, 9, 27, 81, 243]
[1, 3, 9]
[1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683, 59049]

3. (10 points) Modify the example code below to simulate the model given for time steps (step size) of 0.2 and initial conditions of N=100 and P=20. Plot the trajectory as a smooth curve in pink. Plot the vector field for the model given with N ranging between 0 and 150 and P ranging between 0 and 120.

```python
var("N,P")
t = srange(0,100,0.2) # Change step size to 0.2
sol=desolve_odeint([0.5*N - 0.01*N*P, 0.5*0.01*N*P - 0.2*P], ics=[100,20], dvars=[N,P], times=t) # Change initial conditions to N=100 and P=20
list_plot(sol , color = "pink", plotjoined = True, axes_labels = ["N","P"]) # Trajectory - sol is in format (N,P) already
plot_vector_field([0.5*N - 0.01*N*P, 0.5*0.01*N*P - 0.2*P],[N,0 ,150],[P,0 ,120], axes_labels = ["N","P"]) # Plotting vector field
# Remember you can get syntax for plot_vector_field by typing 
```

(N, P)
4. (10 points) Write a script that generates the complementary strand (as a list) to a DNA sequence with the variable name `dna_seq` with any number of nucleotides (A,T,C,G) (A and T are complementary, C and G are complementary). For example, if the script is given `dna_seq = "ACTGACTGAC"`, it should print “TGACTGACTG”. The variable `dna_seq` can be a string or list of characters such as ["A","C","T"] with only A,C,T,G. You don’t need to consider the possibility of other characters.

```python
# Define list of characters called dna_seq
dna_seq = ["A","C","T","G","A","C","T","G","A","C"]  # Define list of characters called dna_seq
# Define string called dna_seq - works either way
dna_seq = "ACTGACTGAC"  # Define string called dna_seq - works either way
comp = []  # Create empty list for complement
for nucleotide in dna_seq:  # Go through dna_seq
```

if nucleotide == 'A': # determine what nucleotide is in each if,
    comp.append('T') # print complementary nucleotide
elif nucleotide == 'T':
    comp.append('A')
elif nucleotide == 'C':
    comp.append('G')
else:
    comp.append('C')
# Print out complement list

# Also works in same way if you use dna_seq = "ACTGACTGAC"
# Would be a really good idea to test this on another string or list
# of characters, so you know it really works!