Name ____________________

Student Number ____________________

MATH 1MP3

DAY CLASS
DURATION OF EXAMINATION: 2 Hours — Benjamin Bolker
MCMASTER UNIVERSITY FINAL EXAMINATION — April 2016

THIS EXAMINATION PAPER INCLUDES 5 PAGES AND 9 QUESTIONS. YOU ARE RESPONSIBLE FOR ENSURING THAT YOUR COPY OF THE PAPER IS COMPLETE. BRING ANY DISCREPANCY TO THE ATTENTION OF YOUR INVIGILATOR.

Special Instructions:

- please *circle your family name* above
- **no** external aids (notes, calculator, etc.)
- This paper must be returned with your answers.

---

1. Suppose you are given a dictionary of the form

```
{'joe':("male",25), 'fred':("male",39), 'susan':("female",20)}
```

where each key is a name and each value is a tuple containing the sex and age of that individual.

a. (3 points) write code to count the number of males between the age of 20 and 30 (inclusive) … (in this example, the correct answer would be 1)
b. (3 points) generalizing your previous answer, write a function `count_dict(d,sex,age_lwr,age_upr)` that returns the number of individuals of a specified sex between the age limits. If `sex` is neither `"male"` nor `"female"` it should raise a `ValueError`.
c. (4 points) suppose now that you have the following type of data instead, where the names are defined in a separate dictionary

```
d = {'joe':25, 'fred':39, 'susan':20}
names = {'joe':"male", 'fred':"male", 'susan':"female"}
```

write a function `count_dict2(d,name_dict,sex,age_lwr,age_upr)` that handles this kind of data to solve the same problem defined above.

2. (6 points) The Bessel function can be defined as

$$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\,\Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha}$$

(Wikipedia)

The factorial (`m!`) and Gamma ($\Gamma(.)$) functions can be imported from `scipy` via

```
from scipy.special import gamma,factorial
```

assuming that these functions have already been imported, write a function `besselJ(x,alpha,k=4)` that returns the (approximation to the) Bessel function computed by summing the terms in the series up to *and including* the $k^{\text{th}}$ term (i.e. $\sum_{m=0}^{k}$). (You can assume that the input is legal, i.e. that `x` is a non-negative floating point number, `nu` is a floating point number, and `k` is an integer.)

3. There is something wrong with each of the following examples: they "should" produce a `True` value, but they don't (they produce either a non-`True` value or an error). State what value/error they produce and give a *short* (one-sentence) explanation what has gone wrong. (2 points each)

a. check that $\left(\sqrt{2}\right)^2 = 2$:

```
import numpy as np
np.sqrt(2)**2==2
```

b. list reversal:

```
def rev(x):
   x.reverse()
   return(x)

L = [0,1,2,3]
L_rev = rev(L)
L[1] == 1
```

c. extract the third element of a list:

```
a = [1,2,3]
a[3] == 3
```

d. compute $\sum_{i=0}^{3} i^2$:

```
for i in range(4):
   k = 0
   k += i**2
k == 14
```

4. Collatz conjecture

a. (6 points) Write a function `def collatz(n,itmax=1000)` that, for any given value of `n`,

- if `n` is even, divide it by 2
- if `n` is odd, multiply it by 3 and add 1

and continues these steps until more than `itmax` steps have been taken *or* `n` is equal to 1. The function should return the total number of times through the cycle. For example, for `collatz(5)`, the sequence would be 5, 16, 8, 4, 2, 1 and the function would return 5. For `collatz(6)` the sequence would be 6, 3, 10, 5, 16, 8, 4, 2, 1 and the function would return 8.

b. (3 points) Using this function, write Python code that computes the number of steps required for each value between 1 and 10000 (inclusive) and saves the results in a `numpy` array (plotting the resulting array would produce the following picture … which is, however, completely irrelevant for the purposes of the exam)

5. (6 points) The function `os.listdir()` returns a list of the names of files found in a directory. Suppose that `L` is the result of this command, and that every file in the directory contains a single column of

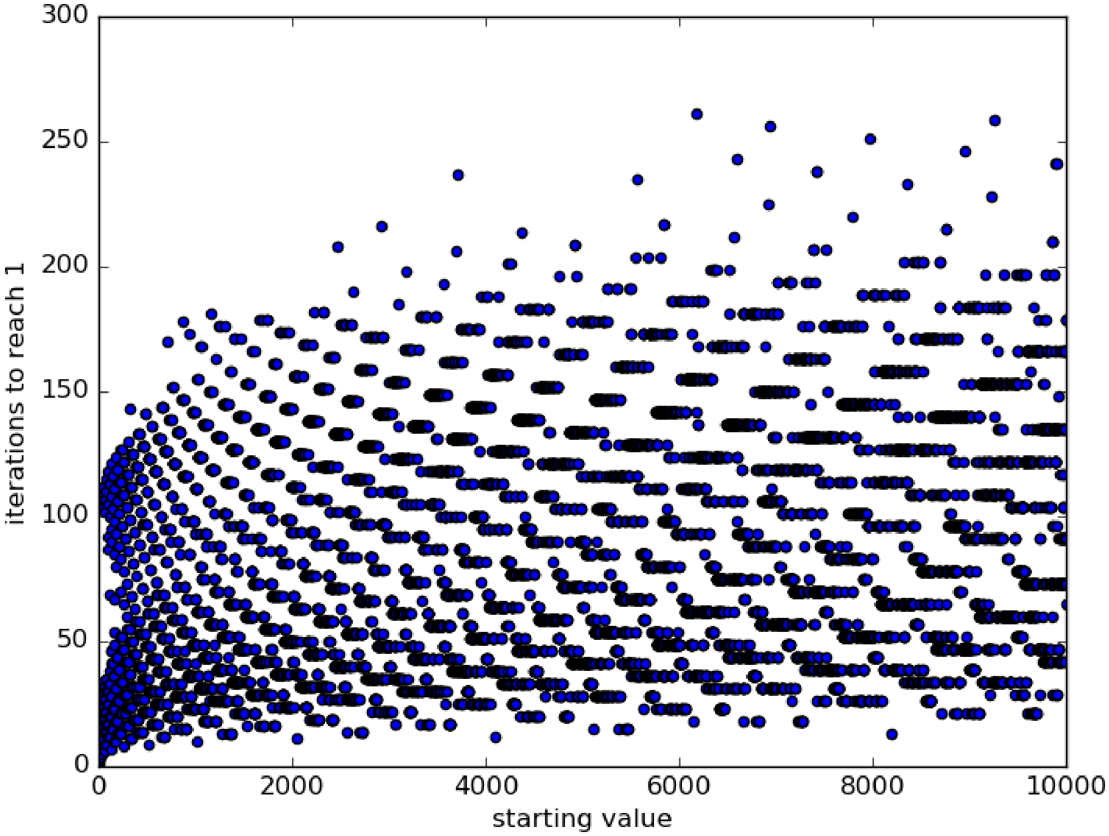
300
250
200
150
100
50
0
iterations to reach 1
0
2000
4000
6000
8000
10000
starting value

Figure 1:

numbers, and that every file has the same number of rows. Write a program that reads each file and combines them into a single `numpy` array *of floats*. Keep in mind that:

- if `fn` is a file name, `open(fn)` opens the file;
- if `f` is an open file, `f.read().split()` will read the entire file and split it on whitespace, returning a list *of characters*:
- `numpy.array` has a `dtype` argument that will convert its argument to the specified type

For example, if there were three files in the directory: `a.txt`, `b.txt`, and `other_file.txt`,

```
a.txt          b.txt          other_file.txt
------         -------        --------------
1              17             4
2              18             5
3              150            6
```

then the result would be

```
[[   1.   17.    4.]
 [   2.   18.    5.]
 [   3.  150.    6.]]
```

6. (5 points) Draw an approximation of the picture that the following code produces. Include x- and y-axis limits.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(10)
fig, ax = plt.subplots() ## open a figure containing a single axis
ax.plot(x,x**2)
ax.scatter(x,-np.sqrt(x))
fig.show()
```

7. (3 points for each item) Given a two-dimensional `numpy` array `a`, write a single line of code **using slicing or ranges** to extract various components. As an example, suppose `a` is of the form

```
  1   2   3 ...   4   5
 17  21  18 ...  90  91
  4   6   9 ...   8   7
... ... ... ... ... ...
 12  17  18 ...  21  22
  2   1   7 ...   3   4
  1   8   9 ...   6   4
```

(where `...` stands for some number of omitted rows/columns)

a. the element in the first row, second column (2 in the example)
b. the third row ([4 6 9 . . . 8 7] in the example)
c. the last column ([5 91 7 . . . 22 4 4] in the example)
d. the last three elements in the last column ([22 4 4] in the example)

8. (3 points for each item) Suppose the file `weather.csv` looks like this:

```
year,month,day,time,temp,wind,wind_dir,precip,precip_type
2014,   01, 01,0800,  -3,   1,      NW,     0,          *
2014,   01, 01,0900,  -2,   0,       *,     0,          *
2014,   01, 01,1000,   0,   0,       *,     2,       snow
2014,   12, 31,1100, -18,   0,       *,     1,       snow
```

Now we run the following `pandas` code:

```
import pandas as pd
dd = pd.read_csv("weather.csv",na_values=["*"])
```

a. what is the value of `dd.loc[2,"temp"]`?
b. what is the value of `dd.iloc[1,6]`? What does this mean?
c. what are the results of running

```
dd2 = dd[(dd.temp<0) & (dd.precip>0)]
print(dd2.precip_type)
```

?

9. Extra credit (3 points)

What is wrong with this code? Why doesn't it return `True`, and what does it do instead?

```
def foo(x):
   return(x.sort())
a = [1,4,9,2]
b = foo(a)
b[3] == 9
```

**The End**