C200 Programming Assignment № 3

Dr. M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

February 11, 2023

Due Date: February 17th 2023 at 11:00 PM. Please start this homework in a day or so. Do not rush solving–it makes for unpleasant experience and interferes with learning. Please submit your work to the Autograder: `https://c200.luddy.indiana.edu/` and push to GitHub before the due date. **Student Pairs** are provided at the end of this document.

Instructions for submitting to the Autograder

1. Make sure that you are **following the instructions** in the PDF, especially the format of output returned by the functions. For example, if a function is expected to return a numerical value, then make sure that a numerical value is returned (not a list or a dictionary). Similarly, if a list is expected to be returned then return a list (not a tuple, set or dictionary).
2. Test **debug** the code well (syntax, logical and implementation errors), before submitting to the Autograder. These errors can be easily fixed by running the code in VSC and watching for unexpected behavior such as, program failing with syntax error or not returning correct output.
3. Given that you already tried points 1-3, if you see that Autograder does not do anything (after you press 'submit') and waited for a while (30 seconds to 50 seconds), try refreshing the page or using a different browser.
4. Once you are done testing your code, comment out the tests i.e. the code under the __name__ == "__main__" section.

Some reminders from lecture

Please read through the problems carefully.

- A **constant** function is a function whose output value is the same for every input value. For example: f(x) = 3, irrespective of what we plug in for x, the function f will allways output 3.
- Now we know that a **constant** (or fixed) function $f(x)$ returns the same constant value, *e.g.*,

$$f(x, y) = 3 \tag{1}$$

  No matter the inputs, the value is three.

- We can convert between $\log_a, \log_k$:

$$\log_b(x) = \frac{\log_k(x)}{\log_k(b)} \tag{2}$$

  Why? Remember that if $\log_b(x) = r$, then $b^r = x$. So, let's first write

$$\log_b(x) = r \tag{3}$$
$$\log_k(x) = s \tag{4}$$
$$\log_k(b) = t \tag{5}$$

  This means

$$b^r = x \tag{6}$$
$$k^s = x \tag{7}$$
$$k^t = b \tag{8}$$

  We see that $b^r = x = k^s$. Since $b = k^t$, we can write:

$$(k^t)^r = k^{rt} = x = k^s \tag{9}$$

  Then

$$\log_k(k^{rt}) = \log_k(x) = \log_k(k^s) \tag{10}$$
$$rt = \log_k(x) = s \tag{11}$$

  Using equations 3,5 for r, t we have

$$\log_b(x)\log_k(b) = \log_k(x) \tag{12}$$
$$\log_b(x) = \frac{\log_k(x)}{\log_k(b)} \tag{13}$$

Problem 1: Functions and math module

All the following functions are drawn from real-world sources. This is an exercise for you to learn how to use a module on your own. From the math module use

- `math.exp(x)` for e^x
- `math.ceil(x)` for $\lceil x \rceil$ rounds x UP to the nearest integer k such that $x \leq k$
- `math.log(x)` for $\log_e(x) = \ln(x)$.

1. According to the Center for Disease Control (CDC) the bacteria *Salmonella* causes about 20K hospitalizations and nearly 400 deaths a year. The formula for how fast this bacteria grows is:

$$N(n_0, m, t) = n_0 e^{mt} \tag{14}$$
$$\tag{15}$$

   where n_0 is the initial number of bacteria, m is the growth rate *e.g.*, 100 per hr, and t is time in hours. Here is how you would calculate the size for an initial colony of 500 that grows at the rate of 100 per hr, for four hours.

$$N(500, 100, 4) = 2.610734844882072 \times 10^{176} \tag{16}$$

2. The number of teeth $N_t(t)$ after t days from incubation for *Alligator mississippiensis* is:

$$N_t(t) = 71.8e^{-8.96e^{-0.0685t}} \tag{17}$$
$$N_t(1000) = \lceil 71.8 \rceil = 72 \tag{18}$$

3. If we want to calculate the work done when an ideal gas expands isothermally (and reversibly) we use for initial and final pressure $P_i = 10\ bar, P_f = 1\ bar$ respectively at $300^o K$. In this problem we are using $\ln$ which is $\log_e$ ('math.log uses base e by default'). Because $\log_e$ is used so often, you'll see it just as often abbreivated as $\ln$.

$$W(P_i, P_f) = RT \ln(P_i/P_f) \tag{19}$$
$$W(10, 1) = \lceil 8.314(300)(\ln 10) \rceil = 5744 \tag{20}$$

   at temperature T (Kelvin) and $R = 8.314\ J/mol$ the universal gas constant.

4. The Wright Brothers are known for their Flyer and its maiden flight. The formula for lift is:

$$L(V, A, C_\ell) = kV^2 AC_\ell \tag{21}$$
$$L(33.8, 512, 0.515) = \lceil 0.0033(33.8)^2(512)0.515 \rceil = 995 \tag{22}$$

where k is Smeaton's Coefficient ($k = 0.0033$ from their wind tunnel), $V = 33.8\ mph$ is relative velocity over the wing, $A = 512\ ft$ area of wing, and $C_\ell = 0.515$ coefficient of lift. The Flyer weighed $600\,lbs$ and Orville was about $145\,lbs$. We can see that the lift was sufficient since $994 > 745$.

Deliverables for Problem 1

- Complete $N(), N_t(), W(), and\ L()$ functions described above.
- Don't forget to round up using $math.ceil(x)$ as indicated.

Problem 2: Quadratic

We saw, and also know from basic algebra that for $ax^2 + bx + c = 0$ the roots (values that make the equation zero) are given by:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{23}$$

The expression $b^2 - 4ac$ is called the discriminant. By looking at the discriminant we can deteremine properties of the roots:

- If $b^2 - 4ac > 0$, then both roots are real
- If $b^2 - 4ac = 0$, then both roots are $-b/2a$
- If $b^2 - 4ac < 0$, then both roots are imaginary

Write a function $q(t)$ that takes a tuple $t = (a, b, c)$ and returns 1 if the roots are real, 0 otherwise:

$$q((a, b, c)) \quad = \quad \begin{cases} 1 & \text{roots are real} \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

For example,

$$q((1, 4, -21)) \quad = \quad \text{True} \tag{25}$$

$$q((3, 6, 10)) \quad = \quad \text{False} \tag{26}$$

$$q((1, 0, -4)) \quad = \quad \text{True} \tag{27}$$

Deliverables for Problem 2

- Review integers and reals.
- You are not allowed to use the module `cmath`.
- Complete the function.

Problem 3: Somethings Are Taxed, Somethings Are Not

Search on this phrase “what food isn’t taxed in indiana”. You’re writing software that allows for customer checkout at a grocery store. A customer will have a have receipt r which is a list of pairs $r = [[i_0, p_0], [i_1, p_1], \ldots, [i_n, p_n]]$ where i is an item and p the cost. You have access to a list of items that are not taxed $no\_tax = [j_0, j_1, \ldots, j_m]$. The tax rate in Indiana is 7%. Write a function amt that takes the receipt and the items that are not taxed and gives the total amount owed. For this function you will have to implement the member function $m(x, lst)$ that returns True if x is a member of lst.

For example, let $r = [[1, 1.45], [3, 10.00], [2, 1.45], [5, 2.00]]$ and $no\_tax = [33, 5, 2]$. Then

$$amt(r, no\_tax) = round(((1.45 + 10.00)1.07 + 1.45 + 2.00), 2) \quad (28)$$

$$= \$15.7 \quad (29)$$

Deliverables for Problem 3

- Complete $m(x, lst)$ function. You must search for x in lst by looping through lst.
- Complete the $amt()$ function.
- You cannot use the Python’s **in** keyword to search for 0s. For example, you can not use

  ```
  if 0 in my_list:
  ```

  If you use Python’s **in** keyword to search for you, you will recieve a 0. Instead, as mentioned in the first bullet point, you should loop through the list and check each value.
- Round the output of amt() function to 2 decimal places.

Problem 4: 2D intersecting lines

A line in 2D Euclidean space can be described by $y = mx + b$ (m and b are the slope and intercept respectively). You are provided with a function that takes two points and returns the line as a dictionary:

```
#INPUT two points in 2D
#OUTPUT a dictionary {'m':m, 'b':b} holding slope & intercept
#values are rounded to two decimal places
def make_line(p0,p1):
    x0,y0,x1,y1 = *p0,*p1
    m = round((y1 - y0)/(x1 - x0),2)
    b = round(y0 - (m*x0),2)
    return {'m':m, 'b':b}
```

We'll assume all lines are functions. You'll implement a function that takes two lines described above and return the point of intersection (x, y). Here is the function on two inputs:

```
p0 = (32,32)
p1 = (29,5)
p2 = (15,10)
p3 = (49,25)
p4 = (15,30)
p5 = (50,15)

l0,l1 = make_line(p0,p1),make_line(p2,p3)
print(intersection(l0,l1))
l0 = make_line(p4,p5)
print(intersection(l0,l1))
```

has output:

```
[30.3, 16.73]
[37.99, 20.11]
```

Deliverables for Problem 4

- Complete the function.
- You cannot use any math module or cmath module function.
- Round the values of slope and intercept to two decimal places.

Problem 5: Means

When analyzing data, we often want to summarize it: make it concise. Each of these functions takes a list of n numbers as nlst. The mean of a list of numbers gives a summary through one number. You're probably aware of the arithmetic mean:

$$\text{arithmetic\_mean}(\text{nlst}) = \frac{(x_0 + x_1 + \cdots + x_{n-1})}{n} \tag{30}$$

For example, the arithmetic mean of [1,2,3] is 2.0.

The geometric mean (usually done with logs) is:

$$\text{geo\_mean}(\text{nlst}) = a^{sum/n} \tag{31}$$

$$sum = \log_a(x_0) + \log_a(x_1) + \cdots + \log_a(x_{n-1}) \tag{32}$$

where $\log_a$ is an arbitrary log to base a. For example, the geometric mean of [2,4,8] is 4.0. Use $\log_{10}$ as default–but it doesn't actually matter. The harmonic mean is:

$$\text{har\_mean}(\text{nlst}) = \frac{n}{1/x_0 + 1/x_1 + \cdots + 1/x_{n-1}} \tag{33}$$

For example, the harmonic mean of [1,2,3] is approximately 1.64.

The root mean square is:

$$\text{RMS\_mean}(\text{nlst}) = \sqrt{\frac{sum}{n}} \tag{34}$$

$$sum = x_0^2 + x_1^2 + \cdots + x_{n-1}^2 \tag{35}$$

For example, the root mean square of [1,3,4,5,7] is approximately 4.47.

All of these functions take a (possibly empty) list of numbers. If there is a list of numbers, then return the mean. If the list is empty, return the string **"Data Error: 0 values"**. If there is a zero in the list of numbers for the harmonic mean, then return the string **"Data Error: 0 in data"**. We give these two errors, because we face division by zero if we don't.

Deliverables for Problem 5

- Complete the functions as specified above.
- You cannot use the Python's **in** keyword to search for 0s.
- In case of empty list or 0 in harmonic mean-return the error string exactly as mentioned in the problem description. Do not add any extra white spaces.

Problem 6: Occurs

In this problem you'll write one function occur_at_least(x,lst,y) that returns True if object x occurs at least y times in lst, False otherwise. We assume lst is either a list or a tuple. Here are a couple of runs:

```
data = [[1,[1,2,1,2,1,1],4], [1,[1,2,1,2,1,1],3],
        [1,(1,2,1,2,1,0),4], ]

for d in data:
    print(occur_at_least(*d))
```

has output:

```
True
True
False
```

Deliverables for Problem 6

- Complete the function.

Problem 7: Mortgage

A *mortgage* is what you pay when you cannot purchase, usually a home, outright. You pay in installments that have to do with *terms* of the agreement. This includes an interest rate that is added to your payments. Here is the formula for your monthly payment:

$$m = P\frac{i(1+i)^n}{(1+i)^n - 1}$$

where P is the cost of the home, i is the percentage over 12 months, n is the total number of months. Let's say our data is: $300,000 house at 2.9% for 30 years. Then

$$\begin{aligned} m &= 300000\frac{.029/12(1+.029/12)^{30\times12}}{(1+.029/12)^{30\times12}-1} \\ &= 300000\frac{.0024166(1.0024166)^{360}}{(1.0024166)^{360}-1} \\ &= 300000\frac{.0024166(2.38440696)}{2.38440696-1} \\ &= 300000\frac{0.005762}{1.38440696} \\ &= 300000(0.004162185) = \$1248.69/mo \end{aligned}$$

The data should be in a list, *i.e.*, house = [300000,2.9,30] which represents the value of the house, the interest rate and the years. If house = [100000,6.0,30], the monthly payment is about $599.95. We will call this function Mortgage (house)

This seems easy financially. You can find what the mortgage *actually* costs by finding what you paid for the house and what its original value was:

$$\$1248.69/mo(30\,yr)(12\ mo/yr) - \$300000$$
$$\$449528.40 - \$300000 \approx \$149528.40$$

We will call this function total_paid(house).

Deliverables for Problem 7

- Round the return value of total_paid() to two decimal places.
- Complete the functions.
- The total_paid function must use the Mortgage function. Both take lists described above.

Problem 8: Geometric Series

A geometric series is the sum of an infinite number of terms that have a constant ratio between successive terms. For example,

$$\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16} \tag{36}$$

is apparently a geometric series since:

$$\frac{1/4}{1/2}=\frac{1/8}{1/4}=\frac{1/16}{1/8}=\tfrac{1}{2} \tag{37}$$

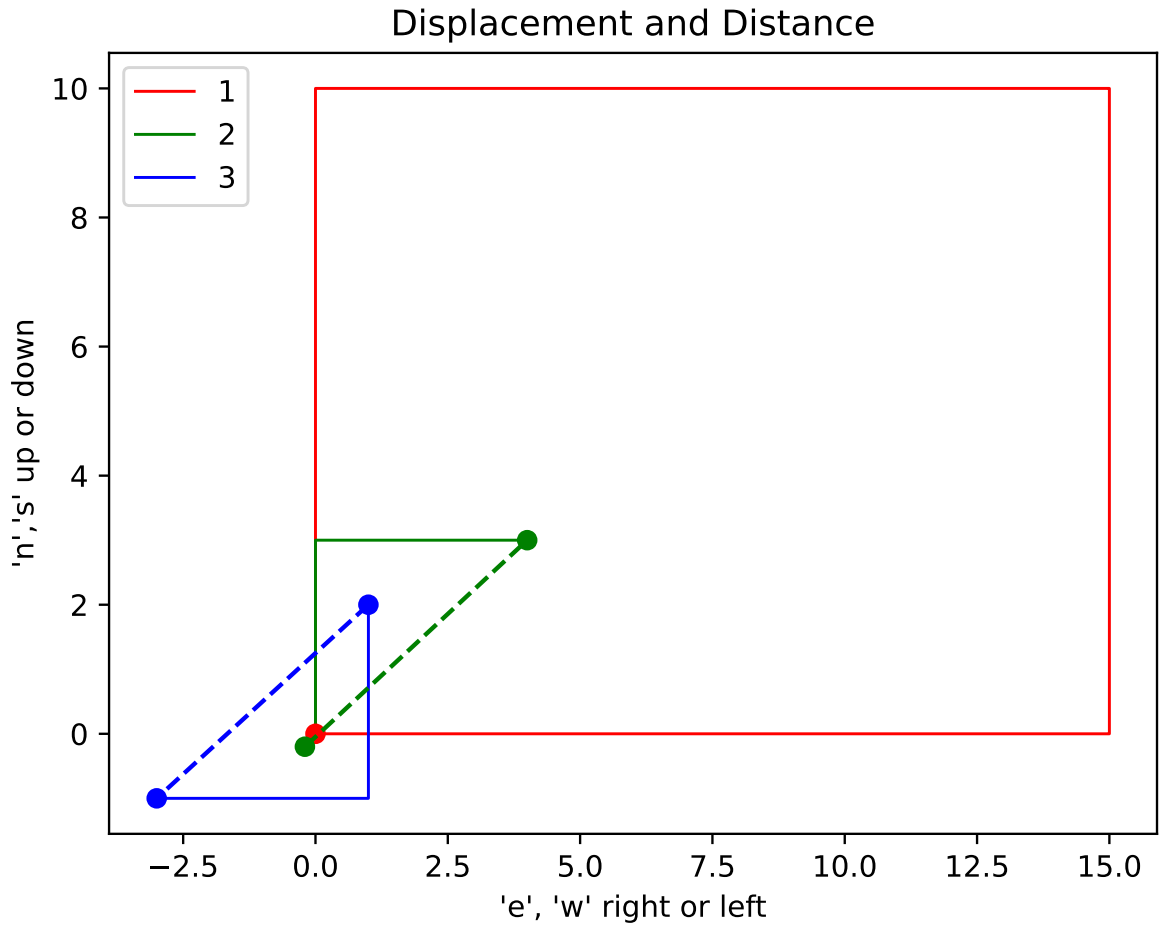
Write a function $is\_geo(xlst)$ that takes a list of numbers and returns 1 only if the series is geometric; otherwise 0. If the list has 2 or fewer members, is_geo returns 0. For example,

$$is\_geo([1/2,1/4,1/8,1/16,1/32]) = 1 \tag{38}$$
$$is\_geo([1,-3,9,-27]) = 1 \tag{39}$$
$$is\_geo([625,125,25]) = 1 \tag{40}$$
$$is\_geo([1/2,1/4,1/8,1/16,1/31]) = 0 \tag{41}$$
$$is\_geo([1,-3,9,-26]) = 0 \tag{42}$$
$$is\_geo([625,125,24]) = 0 \tag{43}$$
$$is\_geo([1/2,1/4]) = 0 \tag{44}$$

Deliverables for Problem 8

- Complete the function
- Assume none of the numbers are zero
- If the series has 2 or fewer the function returns 0

Problem 9: Tracking movement of objects



Each color is a run. Since green and red start at the same location, the green is slightly offset so it is visible. The dots show the starting and ending positions. Blue and green have dashed lines, because their starting and ending points are different. The dashed lines both have size 5.

For this problem, we're tracking movement of objects. This is first step in autonomous vehicle movement. You'll write two functions. The first function determines the distance in 2D space:

$$net\_displacement(p_0, p_1) \quad = \quad [(x_0 - x_1)^2 + (y_0 - y_1)^2]^{1/2}$$

where $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$. The second function called track(start_pos, movement) takes a starting point (x, y) and a list of movements of size one. The list is the first letter of "east", "west", "north", "south". If the current position is (x, y), then each letter changes the values:

'e'	move right (add one to x)
'w'	move left (subtract one from x)
's'	move down (subtract one from y)
'n'	move up (add one to y)

For example, if you're starting at $(3, 0)$ then ['e','e','w','s','s','s'] results in $(4, -3)$. The total distance travelled is 6 (units). The function returns a tuple: the current location, the distance

traveled, and the actual distance between the start and end points. We use net displacement to find how far we are from our starting position. The graphs below show three calls: red, blue, green. Look at the calls of the function with the graphic (built from Python too!). Here are the three calls:

```
data_m = [[(0,0), list(10*'n' + 15*'e' + 10*'s'+15*'w')],
          [(0,0), list(3*'n' + 4*'e')],
          [(1,2), list(3*'s' + 4*'w')]]

for d in data_m:
    print(track(*d))
```

has output:

```
((0, 0), 50, 0.0)
((4, 3), 7, 5.0)
((-3, -1), 7, 5.0)
```

So, the first run started at (0,0), took 50 steps, and ended-up back where it started. To make encoding more convenient, we take advantage of typecasting from strings to lists. For a string “abc...xyz”, list(“abc...xyz”) returns [‘a’,‘b’,‘c’,...,‘x’,‘y’,‘z’].

Deliverables for Problem 9

- Complete the two functions.
- Round the net_displacement to two decimal places.

Problem 10: Distance between two objects

Assume we are tracking two vehicles using the system in problem 9. Write a function final_distance(m0,m1) that takes two tuples (outputs from track) and determines the final distance between the two vehicles. For example,

```
1 data_m = [[(0,0), list(10*'n' + 15*'e' + 10*'s'+15*'w')],
2           [(0,0), list(3*'n' + 4*'e')],
3           [(1,2), list(3*'s' + 4*'w')]]
4
5 print(final_distance(track(*data_m[1]),track(*(data_m[2]))))
```

gives output:

```
1 8.06
```

because

$$net\_displacement((4,3),(-3,-1)) \quad = \quad [(4-(-3))^2+(3-(-1))^2]^{1/2} = [49+16]^{1/2} \approx 8.06$$

As a reminder, the distance between two objects is computed generally as

$$net\_displacement(p_0,p_1) \quad = \quad [(x_0-x_1)^2+(y_0-y_1)^2]^{1/2}$$

Deliverables for Problem 10

- Complete the function.
- While you are free to fully implement this, problem 9 already has the function(s) you need obviating a lot of coding. You can call the same functions to solve this function and only write new code as needed.

Problem 11: Big 10 Women's College Hoops: IU

The IU women's team is ranked 2nd. Tonight they beat Iowa ranked 5th. You'll write a function update(conference, game) that updates the conference standings using the output of the game. A game is a dictionary {team0:score0, team1:score1, ... } where team0,team1 are strings and score0,score1 are non-negative integers. For this problem, you'll have to loop over the elements of the dictionary. To remind you, for a dictionary d

- $d.keys()$ returns an iterable of dictionary keys
- $d.values()$ returns an iterable of dictionary values
- $d.items()$ returns an iterable of tuples that are key,value pairs

Since the dictionary is mutable, if you send it as a parameter and change it in the function, it'll be changed globally. The function does **not** return any value. The function updates the wins, losses ('W','L') and percentage wins: $\frac{wins}{wins+losses}$ ('PCT'), while retaining the number of games played at their home campus versus off-campus ('Home'). Here's a run (some of the data is modified to make the problem easier)

```
1 big_10_women = {'IU':{'W':12,'L':1,'PCT':.923, 'Home':(13,0)},
2                 'PU':{'W':6,'L':6,'PCT':.500, 'Home':(8,4)},
3                 'IOWA':{'W':11,'L':1,'PCT':.917, 'Home':(11,1)},
4                 'NW':{'W':1,'L':11,'PCT':.083,'Home':(6,6)}}
5
6 print(big_10_women['IU'],big_10_women['IOWA'])
7 update(big_10_women,{'IU':87,'IOWA':78})
8 print(big_10_women['IU'],big_10_women['IOWA'])
```

has output:

```
1 {'W': 12, 'L': 1, 'PCT': 0.923, 'Home': (13, 0)} {'W': 11, 'L': 1, 'PCT': ↩
    0.917, 'Home': (11, 1)}
2 {'W': 13, 'L': 1, 'PCT': 0.929, 'Home': (13, 0)} {'W': 11, 'L': 2, 'PCT': ↩
    0.846, 'Home': (11, 1)}
```

You can see we added one to the wins for IU, changed the percentage, added one to the losses for Iowa, and changed the percentage.

Deliverables for Problem 11

- Complete the function.
- Return the dictionary (conference) after updating it with the game info i.e. wins, loses and PCT.

Problem 12: Go Fund Me

In this problem, you'll be writing code that takes donations for a goal amount that is initially posted. The values are assumed to be dollars. The list contains the individual donations. You'll keep taking donations until the goal is met. The function returns a tuple: the first value is the original goal, the second value is the list of possible donations that are left if the goal is minimally met, and the last value is the amount that is needed to meet the goal. If the goal is not met, the last value returned should be negative. For this problem, using enumerate is very useful. Here are a couple of runs:

```
data = [[100,[10,15,20,30,29,13,15,40]],
        [100,[]],
        [100,[30,4]]]

for d in data:
    print(go_fund_me(*d))
```

has output:

```
(100, [13, 15, 40], 4)
(100, [], -100)
(100, [], -66)
```

The first tuple says the goal of $100 has been met with 10+15+20+30+29 = 104. No more donations are needed. The remainder [13,15,40] are returned to the people and 4 is the amount over. For the second donation, nobody donated so the goal remains unchanged. For the third, only two people donated. The goal that's left is $66.

Deliverables for Problem 12

- Complete the function.

Student Pairs

aaberma@iu.edu, sihamza@iu.edu
actonm@iu.edu, arjbhar@iu.edu
adagar@iu.edu, coldjone@iu.edu
brakin@iu.edu, aguarda@iu.edu
sakinolu@iu.edu, mostrodt@iu.edu
moalnass@iu.edu, quecox@iu.edu
anderblm@iu.edu, ajneel@iu.edu
jaybaity@iu.edu, klongfie@iu.edu
nbakken@iu.edu, criekki@iu.edu

chnbalta@iu.edu, ekkumar@iu.edu
nokebark@iu.edu, hvelidi@iu.edu
nwbarret@iu.edu, patel88@iu.edu
sbehman@iu.edu, nbulgare@iu.edu
jadbenav@iu.edu, ghyatt@iu.edu
dombish@iu.edu, chrinayl@iu.edu
gavbish@iu.edu, hollidaa@iu.edu
sebisson@iu.edu, kynogree@iu.edu
aibitner@iu.edu, yijwei@iu.edu
abolad@iu.edu, arykota@iu.edu
jacbooth@iu.edu, jarymeln@iu.edu
joshbrin@iu.edu, wdoub@iu.edu
neybrito@iu.edu, psaggar@iu.edu
ivycai@iu.edu, greymonr@iu.edu
bcarl@iu.edu, peschulz@iu.edu
ecastano@iu.edu, chnico@iu.edu
cheng47@iu.edu, mgambett@iu.edu
kbchiu@iu.edu, roelreye@iu.edu
hc51@iu.edu, jmom@iu.edu
joracobb@iu.edu, zisun@iu.edu
sgcolett@iu.edu, wallachl@iu.edu
swcolson@iu.edu, lsherbst@iu.edu
joecool@iu.edu, jegillar@iu.edu
jacosby@iu.edu, voram@iu.edu
jacuau@iu.edu, schwani@iu.edu
aadidogr@iu.edu, hnichin@iu.edu
siqidong@iu.edu, jl263@iu.edu
dud@iu.edu, omilden@iu.edu
marbelli@iu.edu, dwinger@iu.edu
moesan@iu.edu, webejack@iu.edu
abdufall@iu.edu, alescarb@iu.edu
stfashir@iu.edu, rpmahesh@iu.edu
sfawaz@iu.edu, esmmcder@iu.edu
jfearri@iu.edu, caldsmit@iu.edu
tyfeldm@iu.edu, clschel@iu.edu
tflaa@iu.edu, eddykim@iu.edu
lgflynn@iu.edu, vilokale@iu.edu
megofolz@iu.edu, jmissey@iu.edu
bgabbert@iu.edu, rythudso@iu.edu
mdgamble@iu.edu, saseiber@iu.edu
ogift@iu.edu, danwils@iu.edu

egillig@iu.edu, aalesh@iu.edu
agoldswo@iu.edu, sijatto@iu.edu
mgorals@iu.edu, bensokol@iu.edu
davgourl@iu.edu, nklos@iu.edu
qugriff@iu.edu, ss126@iu.edu
jonhick@iu.edu, swa5@iu.edu
achimeba@iu.edu, cjromine@iu.edu
anthoang@iu.edu, cartmull@iu.edu
qhodgman@iu.edu, avpeda@iu.edu
tifhuang@iu.edu, georliu@iu.edu
huntbri@iu.edu, jvalleci@iu.edu
bradhutc@iu.edu, fshamrin@iu.edu
jackssar@iu.edu, hmerrit@iu.edu
ajarillo@iu.edu, parkecar@iu.edu
ljianghe@iu.edu, jcpilche@iu.edu
sjkallub@iu.edu, jyamarti@iu.edu
dilkang@iu.edu, trewoo@iu.edu
pkasarla@iu.edu, bpoddut@iu.edu
katzjor@iu.edu, sl92@iu.edu
aketcha@iu.edu, jl335@iu.edu
dk80@iu.edu, rair@iu.edu
delkumar@iu.edu, mszczas@iu.edu
akundur@iu.edu, grschenc@iu.edu
aladkhan@iu.edu, sousingh@iu.edu
jolawre@iu.edu, mmandiwa@iu.edu
jacklian@iu.edu, sjvaleo@iu.edu
lopeal@iu.edu, elyperry@iu.edu
dloutfi@iu.edu, zhaozhe@iu.edu
jasoluca@iu.edu, pvinod@iu.edu
imalhan@iu.edu, davthorn@iu.edu
lmamidip@iu.edu, cmulgrew@iu.edu
cmarcucc@iu.edu, mw154@iu.edu
kalomart@iu.edu, tshore@iu.edu
crfmarti@iu.edu, svaidhy@iu.edu
rmatejcz@iu.edu, sowmo@iu.edu
imaychru@iu.edu, wuyul@iu.edu
nkmeade@iu.edu, dannwint@iu.edu
fimitch@iu.edu, bashih@iu.edu
almoelle@iu.edu, edshipp@iu.edu
nmonberg@iu.edu, mvanworm@iu.edu
isarmoss@iu.edu, jorzhang@iu.edu

masmuell@iu.edu, leplata@iu.edu
ornash@iu.edu, vthakka@iu.edu
snuthala@iu.edu, aw149@iu.edu
sabola@iu.edu, yasmpate@iu.edu
lenonti@iu.edu, dpepping@iu.edu
parisbel@iu.edu, macsvobo@iu.edu
arnpate@iu.edu, kvanever@iu.edu
btpfeil@iu.edu, alexschu@iu.edu
nireilly@iu.edu, caitreye@iu.edu
shahneh@iu.edu, dy11@iu.edu
shethsu@iu.edu, yz145@iu.edu
gsprinkl@iu.edu, sjxavier@iu.edu
marystre@iu.edu, cartwolf@iu.edu
ttieu@iu.edu, jonvance@iu.edu
johwarre@iu.edu, jwescott@iu.edu