C200 Programming Assignment № 4

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

Indiana University, Bloomington, IN, USA

February 17, 2023

Introduction

Due Date: 10:59 PM, Friday, February 24th 2023 Submit your work to the Autograder `https://c200.luddy.indiana.edu/` and remember to add, commit and push to GitHub **before the deadline. We do not accept late submissions.**

In this homework, you'll start mastering your skill in writing functions. Make sure to start working early on this assignment. If we do not explictly mentioned a function/library to use, you are not allowed to use it. For example, if we have not mentioned to use tkinter library then you should not import tkinter in your python file.

Instructions

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. Make sure that the **code does not have infinite loop (that never exits) or an endless recursion (that never completes)** before submitting to the Autograder. You can easily check for this by running in VSC and watching for program output, if it terminates timely or not.
4. 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.
5. Once you are done testing your code, comment out the tests i.e. the code under the __name__ == "__main__" section.

Problem 1: Day of the Week

The modulus operator is denoted by the symbol %. For $x\%y$ it returns the remainder after y is divided into x a whole number of times. While you've seen this before, the format is likely different. To refresh your memory you can visit `https://realpython.com/python-modulo-operator/` or do your own search. For this problem, you will also need to use the floor function in python - think of floor as a way to round down a floating point number to the nearest integer. For example, floor(18.90) is 18 and floor(14.25) is 14.

An approximate (works generally okay for a number of dates, but **not** all) formula for computing the day of the week given dlst = [d,m,y] where d is day, m is month, and y is year is:

$$
\begin{aligned}
dlst &= [d, m, y] && (1)\\
a(dlst) &= y - \frac{(14-m)}{12} && (2)\\
x &= a(dlst) + \frac{a(dlst)}{4} - \frac{a(dlst)}{100} + \frac{a(dlst)}{400} && (3)\\
b(dlst) &= floor(x) && (4)\\
c(dlst) &= m + 12(\frac{14-m}{12}) - 2 && (5)\\
day((d,m,y)) &= (d + b(dlst) + (31 \cdot \frac{c(dlst)}{12}))\%7 && (6)
\end{aligned}
$$

The function day returns a number $1, 2, \ldots, 7$ where $1 = Monday, 2 = Tuesday, \ldots$. If we use this number as the key with the week dictionary, we are able to return the correct day of the week. Here is the dictionary used:

```python
week = {1:"Mon", 2:"Tue", 3:"Wed", 4:"Thu", 5:"Fri", 6:"Sat", 7:"Sun"}
```

For example,

$$
\begin{aligned}
day([14,2,2000]) &= \text{Mon} && (7)\\
print(day([14,2,1963])) &= \text{Thu} && (8)\\
print(day([14,2,1972])) &= \text{Mon} && (9)
\end{aligned}
$$

2/14/2000 falls on a Monday; 2/14/1963 falls on a Thursday; 2/14/1972 falls on a Monday.

Deliverables for Problem 1

- Complete the functions described above.
- For calculating b(dlst), you can use the math.floor() function from the math library.
- Remember, the day function utilizes the week dictionary.

Problem 2: Starting quantum computing and quantitative finance

Quantum Computing is a different model of computation imagined by the physicist Feynman. Instead of whole numbers, like we use in the Turing model, it uses complex numbers. Complex numbers, if you remember, are actually pairs of real numbers written as:

$$\text{complex number} \quad = \quad x \pm y\, i \tag{10}$$

where $x, y \in \mathbb{R}$ (they're just numbers) and there's a lone $i = \sqrt{-1}$. The x is called the real part and the y is called the imaginary part. For example,

$$x^2 + 1 \quad = \quad 0 \tag{11}$$

$$x \quad = \quad \sqrt{-1} = i \tag{12}$$

Then

$$x^2 + 1 \quad = \quad (\sqrt{-1})^2 + 1 = -1 + 1 = 0 \tag{13}$$

In Python we use the function complex(x,y) to form a complex number $(x \pm yj)$ where j represents i and there's no space between the $\pm$ sign and numbers. Let's write the function on line (10), build the complex number, and show it's a solution.

```
>>> def f(x):
...     return x**2 + 1
...
>>> root = complex(0,1)
>>> f(root)
0j
>>> f(root) == 0
True
>>> f(root).real
0.0
>>> f(root).imag
0.0
```

For a quadratic from last homework, you know that the answer is either real or complex. To remind you, the answer is complex when the discriminant is negative. Fig. 1. visualizes what's happening–the curve for complex roots does **intersect** the abscissa. Observe that you'll get zero (on the x-axis or *abscissa*) if you put either of these two x values there. Sometimes the discriminant (the value in the squareroot) is negative. This is where i comes in. We simply multiply the value by -1, thereby allowing us to take the squareroot, but then we append an i to signal it's imaginary.

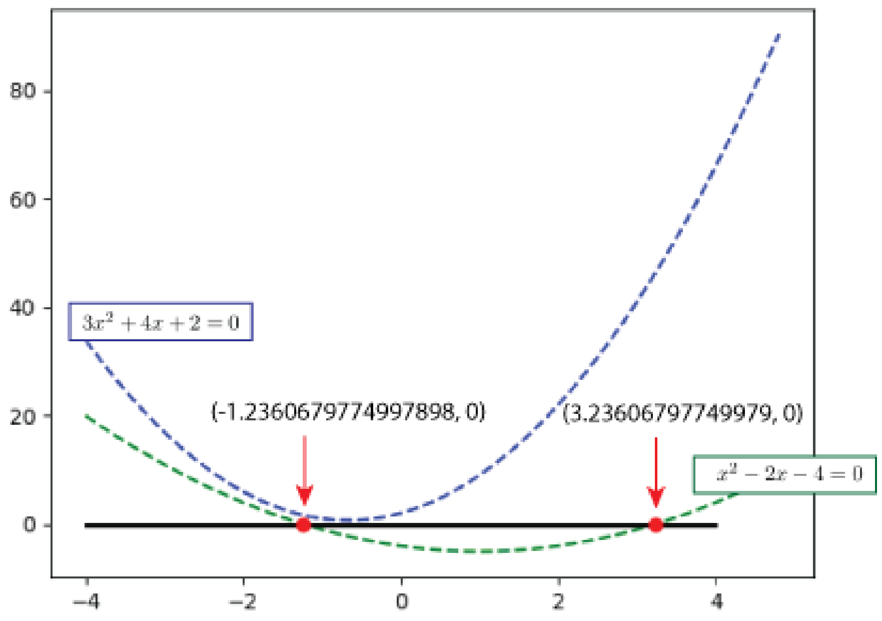

Figure 1: The red dots (with red arrows) are where your solutions are to $x^2 - 2x - 4 = 0$. The dash curve is the function itself. The horizontal line just makes it easier to see the x-axis. The two values are -1.2360679774997898 and 3.23606797749979. For the other function $3x^2 + 4x + 2 = 0$ shown in blue, because it has an imaginary part, it cannot cross the axis. You'll use the matplotlib library you were introduced last homework to actually plot this!

For example, suppose we have $3x^2 + 4x + 2 = 0$. Then we find, after some algebra:

$$\begin{aligned}
x &= \frac{-4 \pm \sqrt{-8}}{6} && (14)\\
&= \frac{-4 \pm \sqrt{-2 \times 4}}{6} && (15)\\
&= \frac{-4 \pm \sqrt{-2} \times \sqrt{4}}{6} && (16)\\
&= \frac{-4 \pm 2\sqrt{-2}}{6} && (17)\\
&= -\tfrac{2}{3} \pm \tfrac{\sqrt{2}}{3} i && (18)\\
&= (-\tfrac{2}{3} + \tfrac{\sqrt{2}}{3} i, -\tfrac{2}{3} - \tfrac{\sqrt{2}}{3} i) && (19)
\end{aligned}$$

In Python we would write:

```
>>> import math
>>> complex(-2/3,math.sqrt(2)/3),complex(-2/3,-math.sqrt(2)/3)
((-0.6666666666666666+0.4714045207910317j), ↩
    (-0.6666666666666666-0.4714045207910317j))
>>> x = round(-2/3,2)
>>> y = round(math.sqrt(2)/3)
>>> complex(x,y), complex(x,-y)
((-0.67+0j), (-0.67+0j))
>>> def f(x):
...     return 3*(x**2) + 4*x + 2
```

```
...
>>> f(complex(x,y))
(0.6667000000000001+0j)
>>> f(complex(x,-y))
(0.6667000000000001+0j)
>>> f(complex(-2/3,math.sqrt(2)/3))
0j
```

Observe the difference between using the full root and only to two decimal places. You'll write a function q(t) where:

$$q((a,b,c)) = \begin{cases} (r_0, r_1) & \text{real roots if } b^2 - 4ac \geq 0 \\ (c_0, c_1) & \text{complex roots if } b^2 - 4ac < 0 \end{cases} \tag{20}$$

When returning the roots, round to two decimal places, *i.e.*, round(x,2). Here are a few examples:

$$q((3,4,2)) = ((-0.67 + 0.47j), (-0.67 - 0.47j)) \tag{21}$$
$$q((1,3,-4)) = (-4.0, 1.0) \tag{22}$$
$$q((1,-2,-4)) = (-1.24, 3.24) \tag{23}$$

Deliverables for Problem 2

- Complete the function according to the equations on line 20.
- Don't forget to round to two decimal places.

Problem 3: Computing Relationships

To match people, companies use trigonometry. Assume you have a list of people $[p_0, p_1, \ldots, p_m]$, we find the two different people who have the smallest angle. The way we calculate this is to assume each person is a list (mathematically vector) of 0s and 1s. We need three basic functions: inner product, magnitude, and $\cos^{-1}$. We assume two lists $x = [x_0, x_1, \ldots, x_n], y = [y_0, y_1, \ldots, y_n]$ of 0s and 1s of the same length:

$$inner\_prod(x, y) = x_0y_0 + x_1y_1 + \cdots + x_ny_n \quad (24)$$

$$mag(x) = \sqrt{inner\_prod(x, x)} \quad (25)$$

For the last function (where we calculate the angle), we know that:

$$\cos(\theta) = \frac{inner\_prod(x, y)}{mag(x)mag(y)} \quad (26)$$

In class we learned that we can invert a function and the inverted function is denoted as f^{-1}. So we can:

$$\cos^{-1}(\cos(\theta)) = \cos^{-1}(\frac{inner\_prod(x, y)}{mag(x)mag(y)}) \quad (27)$$

$$\theta = \cos^{-1}(\frac{inner\_prod(x, y)}{mag(x)mag(y)}) \quad (28)$$

The math module has math.acos() for $\cos^{-1}()$. Further, Python returns θ in radians. We have:

$$\pi \text{ radians} = 180 \text{ degrees} \quad (29)$$

Thus, to convert from radians to degree, you **must** multiply your answer by $\frac{180}{\pi}$.

Your task is to first complete the functions “inner_prd“, “mag“ and “angle“ and then use them to write the “match“ and “best_match“ functions. The match function takes a list of people pi where each person is a list of 0s 1s, and returns all unique pairs with the angle in degrees. Note that “match“ returns the output in the format: [[person 1, person2, angle], [person2, person3, angle], [person1, person3, angle]], where each person i.e., person1, person2-is also a list for example, [1,1,1] or [0,1,0]. So it returns a list of lists. For “angle“ function-round your answer to 2 decimal digits.

```python
people0 = [[0,1,1],[1,0,0],[1,1,1]]
print(match(people0))
print(best_match(match(people0)))
```

gives an output

```
[[[0, 1, 1], [1, 0, 0], 90.0],
 [[0, 1, 1], [1, 1, 1], 35.26],
 [[1, 0, 0], [1, 1, 1], 54.74]]
([0, 1, 1], [1, 1, 1], 35.26)
```

We're displaying the result of match so you can see the structure. Each unique pair as an angle. For example:

$$\begin{aligned} inner\_prod([1,0,0],[1,1,1]) &= 1(1)+0(1)+0(1)=1 && (30)\\ mag([1,0,0]) &= \sqrt{inner\_prod([1,0,0],[1,0,0])}=\sqrt{1}=1 && (31)\\ mag([1,1,1]) &= \sqrt{inner\_prod([1,1,1],[1,1,1])}=\sqrt{3} && (32)\\ \theta &= (\frac{180}{\pi})\text{math.acos}(\frac{1}{1\sqrt{3}}) \approx 54.74 && (33)\end{aligned}$$

Deliverables for Problem 3

- Complete the functions.
- Keep in mind that function angle() utilizes equation 28.
- Round the output of angle to two decimal places.

Problem 4: Completing the Square

When the roots of a quadratic are real, we can transform the quadratic $ax^2+bx+c=0$ into a simpler form and find the roots more easily:

$$ax^2+bx+c=(x+m)^2+n$$

To find the roots then:

$$\begin{aligned} (x+m)^2+n &= 0\\ (x+m)^2 &= -n\\ x+m &= \pm\sqrt{-n}\\ x &= -m\pm\sqrt{-n}\end{aligned}$$

Through some simple algebra we find that

$$\begin{aligned} m &= \frac{b}{2a}\\ n &= c-\frac{b^2}{4a}\end{aligned}$$

We can then call our transform function that takes a,b,c and returns m,n as seen below:

```python
def c_s(coefficients):
    pass

def q_(coefficients):
    m,n = c_s(coefficients)
    pass
```

Deliverables for Problem 4

- Complete the functions.
- Round the output from c_s() and q_() function to 2 decimal places.
- The q_ function must call the c_s function.

Problem 5: Toward Statistical Analysis

In this problem, you'll write fundamental statistical functions. We assume a list of numbers $lst = [x_0, x_1, \ldots, x_n]$.

$$\begin{aligned} mean(lst) &= (x_0 + x_1 + \cdots + x_n)/\text{len}(lst) && (34)\\ \mu &= mean(lst) && (35)\\ var(lst) &= \frac{1}{\text{len}(lst)}((x_0-\mu)^2 + (x_1-\mu)^2 + \cdots + (x_n-\mu)^2) && (36)\\ std(lst) &= \sqrt{var(lst)} && (37) \end{aligned}$$

For example, $lst = [1, 3, 3, 2, 9, 10]$, rounding to two places

$$\begin{aligned} mean(lst) &= 4.67 && (38)\\ var(lst) &= 12.22 && (39)\\ std(lst) &= 3.5 && (40) \end{aligned}$$

The last function mean_centered takes a list of numbers $lst = [x_0, x_1, \ldots, x_n]$ and returns a new list $[x_0 - \mu, x_1 - \mu, \ldots, x_n - \mu]$. An interesting feature of the mean-centered list is that if you try to calculate its mean, it's zero:

$$\begin{aligned} \mu &= (x_0 + x_1 + \cdots + x_n)/n && (41)\\ lst &= [x_0 - \mu, x_1 - \mu, \ldots x_n - \mu] && (42)\\ mean(lst) &= ((x_0 - \mu) + \cdots (x_n - \mu))/n && (43)\\ &= ((x_0 + \ldots + x_n) + n\mu)/n && (44)\\ &= \mu - \mu = 0 && (45) \end{aligned}$$

Using the same list we have

$$mean(mean\_centered(lst)) = -0.0 = 0 \qquad (46)$$

Deliverables for Problem 5

- Complete the functions.
- Round-up to 2 decimal places the ouptut of mean, var and std functions.

Problem 6: Market Equilibrium

When modeling market behavior we use two curves to indicate supply and demand. You can also think of supply as quantity and demand as want. When the two curves intersect (the common point where they cross), see Fig. 2, there is a point that satisfies both equations. Review the problem of intersecting two lines. Here we have specifically two quadratic functions. Assume

$$s(x) = -.025x^2 - .05x + 60 \tag{47}$$
$$d(x) = 0.02x^2 + .6x + 20 \tag{48}$$

for supply s and demand d. We have a function $equi$ that takes the coefficients of s, d and returns the solution:

$$equi((-.025, -.5, 60), (0.02, .6, 20)) = (20.0, -44.44) \tag{49}$$

Our solution returns the roots to a quadratic equation. Since s, d models the presence of items, only 20 makes sense. HINT: use your q function in Problem 2.

Deliverables for Problem 6

- Complete the function.
- Use q in Problem 2 to make the solution very quick.

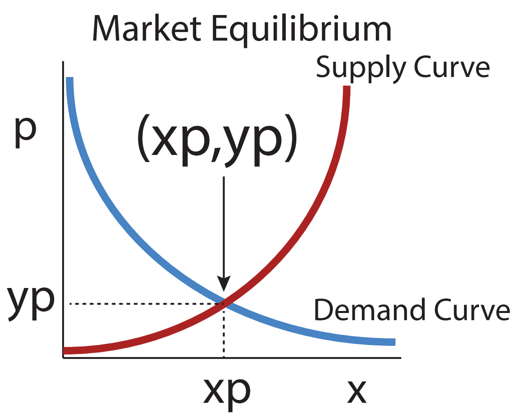

Figure 2: Market equilibrium with supply and demand

Problem 7: Random Selection

We saw in lecture random's choice function. We'll build a better one. In this problem you'll be implementing a function sample(lst,n, seed) that returns n random members from the lst. You'll only be able to use rn.randint(x,y). For example,

```python
lst = [1,2,1,3,4]
print(sample(lst, 3, 1729))
print(sample(lst, 3, 1629))
print(sample(lst, 10, 1729))
```

has output:

```
[1, 3, 4]
[4, 3, 4]
[1, 3, 4, 3, 2, 1, 2, 1, 4, 1]
```

Of course your output will differ if you change the value of seed.

Deliverables for Problem 7

- Complete the function.
- You are only allowed to use rn.randint(x,y).

Problem 8: Counts of Substring

We know that a string in Python is any number of characters enclosed by a set of double or single quotes: Given a string s a substring t is an any slice $t = s[i : j] \; \wedge \; len(t) \neq 0$ for non-

“abcabc”
“”
“ccccc”

negative integers i, j. For the initial strings the counts are shown below. Implement sub_string

“abcabc”	’a’, ’ab’, ’abc’, ’abca’, ’abcab’, ’abcabc’, ’b’, ’bc’, ’bca’, ’bcab’, ’bcabc’, ’c’, ’ca’, ’cab’, ’cabc’
“”	no substrings
“ccccc”	’c’, ’cc’, ’ccc’, ’cccc’, ’ccccc’

to take a string and return a dictionary of counts of all the substrings. For example:

“abc”	’a’: 1, ’ab’: 1, ’abc’: 1, ’b’: 1, ’bc’: 1, ’c’: 1
“”	
“ccccc”	’c’: 5, ’cc’: 4, ’ccc’: 3, ’cccc’: 2, ’ccccc’: 1

Deliverables for Problem 8

- Complete the function.
- You cannot use other string functions other than slicing (which is a strong hint).

Problem 9: Using Loop to Calculate Sinking Fund Schedule

A **sinking fund** is an financial instrument to discharge a future debt. For example, a manufacturing company expects to replace obsolescent machinary in some number of years. In this problem, you are building your finance company to compute payment schedules for a sinking fund. We use the general formula for annuity:

$$\begin{aligned} S &= R\frac{(1+i)^n - 1}{i} \\ i &= \frac{r}{m} \\ n &= my \end{aligned}$$

where S is the future debt, r the (percentage) interest rate, m the number of payments per year, y the number of years, n is the total number of payments (number of payments in a year multiplied by total number of years) and R is the monthly deposit made. Here is the particular problem. Acme Sundials believes replacing their sun dial mold-press will cost $30,000 in two years. They can currently get 10% interest compounded quarterly (four times a year). You build a Python program that, given the initial data will:

- Determine the monthly deposit (R).
- Determine the total schedule of payments that includes the interest accrued and total amount. This should be stored as a list of lists (see example output below).

Here is the run:

```python
def deposit(S,r,i,n):
   pass

def sinking_fund(final_amt,r,m,y):
    pass

for i in sinking_fund(30000,.1,4,2):
    print(i)
```

with output:

```
[[0, 3434.02, 0, 3434.02],
[1, 3434.02, 85.85, 6953.89],
[2, 3434.02, 173.85, 10561.76],
[3, 3434.02, 264.04, 14259.82],
[4, 3434.02, 356.5, 18050.34],
[5, 3434.02, 451.26, 21935.62],
[6, 3434.02, 548.39, 25918.03],
[7, 3434.02, 647.95, 30000.0]]
```

- The function deposit returns the monthly deposit needed for the fund (R).
- Clearly you must solve for R (monthly deposit) first and use this value in sinking_fund() to solve for S.
- Once you have the deposit value, you should find n and i.
- After you have all the values, i..e, i, n, R, you can start calculating your payment schedule.
- Make sure that the payment schedule is stored as a list of lists where each sublist represent one payment as follows: [period, deposit, interest, total fund].

Observe there are eight payments. On the last payment, the fund is \$30,000. The first value in the list is the period, the second is the deposit amount, the third is the interest accrued on the **previous** fund, the last value is the curent total fund. Let's construct period 1 from 0. The total fund in period 0 is \$3434.02, or the initial deposit value for this particular example. Then we have interest:

$$\begin{aligned} i \times fund &= \frac{r}{m} \times 3434.02 \\ &= \frac{.10}{4} \times 3434.02 = 0.025(3434.02) = 85.85 \end{aligned}$$

The total t_1 for that period is the previous total, plus interest, plus deposit:

$$t_1 = 3434.02 + 85.85 + 3434.02 = 6953.89$$

Deliverables for Problem 9

- Complete the functions deposit and sinking_fund.
- We are rounding to two decimal places all values.
- You are free to use a single bounded (hint), unbounded, nested loops–whatever you find most comfortable.

Partners for Programming

aaberma@iu.edu, parisbel@iu.edu
actonm@iu.edu, yz145@iu.edu
adagar@iu.edu, pvinod@iu.edu
brakin@iu.edu, esmmcder@iu.edu
sakinolu@iu.edu, bcarl@iu.edu
moalnass@iu.edu, hnichin@iu.edu
anderblm@iu.edu, rythudso@iu.edu
jaybaity@iu.edu, joshbrin@iu.edu
nbakken@iu.edu, bgabbert@iu.edu
chnbalta@iu.edu, dwinger@iu.edu
nokebark@iu.edu, jmom@iu.edu
nwbarret@iu.edu, swcolson@iu.edu
sbehman@iu.edu, webejack@iu.edu
jadbenav@iu.edu, sl92@iu.edu
arjbhar@iu.edu, dud@iu.edu
dombish@iu.edu, mostrodt@iu.edu
sebisson@iu.edu, kalomart@iu.edu
aibitner@iu.edu, sgcolett@iu.edu
abolad@iu.edu, dk80@iu.edu
jacbooth@iu.edu, bradhutc@iu.edu
neybrito@iu.edu, aadidogr@iu.edu
nbulgare@iu.edu, saseiber@iu.edu
ivycai@iu.edu, ekkumar@iu.edu
ecastano@iu.edu, jmissey@iu.edu
cheng47@iu.edu, mw154@iu.edu
kbchiu@iu.edu, hmerrit@iu.edu
hc51@iu.edu, johwarre@iu.edu
joracobb@iu.edu, qhodgman@iu.edu
joecool@iu.edu, hvelidi@iu.edu
jacosby@iu.edu, zisun@iu.edu
quecox@iu.edu, cjromine@iu.edu
jacuau@iu.edu, marbelli@iu.edu
siqidong@iu.edu, roelreye@iu.edu
wdoub@iu.edu, vilokale@iu.edu
moesan@iu.edu, nklos@iu.edu
abdufall@iu.edu, jorzhang@iu.edu
stfashir@iu.edu, psaggar@iu.edu
sfawaz@iu.edu, jolawre@iu.edu
jfearri@iu.edu, dloutfi@iu.edu

tyfeldm@iu.edu, omilden@iu.edu
tflaa@iu.edu, cmulgrew@iu.edu
lgflynn@iu.edu, qugriff@iu.edu
megofolz@iu.edu, anthoang@iu.edu
mgambett@iu.edu, jyamarti@iu.edu
mdgamble@iu.edu, sabola@iu.edu
ogift@iu.edu, leplata@iu.edu
jegillar@iu.edu, danwils@iu.edu
egillig@iu.edu, mmandiwa@iu.edu
agoldswo@iu.edu, aw149@iu.edu
mgorals@iu.edu, jacklian@iu.edu
davgourl@iu.edu, bensokol@iu.edu
kynogree@iu.edu, isarmoss@iu.edu
aguarda@iu.edu, greymonr@iu.edu
sihamza@iu.edu, alescarb@iu.edu
lsherbst@iu.edu, lmamidip@iu.edu
jonhick@iu.edu, chnico@iu.edu
achimeba@iu.edu, mvanworm@iu.edu
hollidaa@iu.edu, masmuell@iu.edu
tifhuang@iu.edu, svaidhy@iu.edu
huntbri@iu.edu, vthakka@iu.edu
ghyatt@iu.edu, dilkang@iu.edu
jackssar@iu.edu, voram@iu.edu
ajarillo@iu.edu, macsvobo@iu.edu
sijatto@iu.edu, jasoluca@iu.edu
ljianghe@iu.edu, tshore@iu.edu
coldjone@iu.edu, kvanever@iu.edu
sjkallub@iu.edu, aladkhan@iu.edu
pkasarla@iu.edu, rmatejcz@iu.edu
katzjor@iu.edu, bashih@iu.edu
aketcha@iu.edu, patel88@iu.edu
eddykim@iu.edu, jcpilche@iu.edu
arykota@iu.edu, mszczas@iu.edu
delkumar@iu.edu, yasmpate@iu.edu
akundur@iu.edu, shahneh@iu.edu
jl335@iu.edu, cmarcucc@iu.edu
aalesh@iu.edu, klongfie@iu.edu
jl263@iu.edu, crfmarti@iu.edu
georliu@iu.edu, nkmeade@iu.edu
lopeal@iu.edu, jarymeln@iu.edu
rpmahesh@iu.edu, dpepping@iu.edu

imalhan@iu.edu, arnpate@iu.edu
imaychru@iu.edu, cartmull@iu.edu
fimitch@iu.edu, ajneel@iu.edu
almoelle@iu.edu, lenonti@iu.edu
nmonberg@iu.edu, ss126@iu.edu
ornash@iu.edu, sowmo@iu.edu
chrinayl@iu.edu, ttieu@iu.edu
snuthala@iu.edu, grschenc@iu.edu
parkecar@iu.edu, trewoo@iu.edu
avpeda@iu.edu, caitreye@iu.edu
elyperry@iu.edu, sousingh@iu.edu
btpfeil@iu.edu, gsprinkl@iu.edu
bpoddut@iu.edu, rair@iu.edu
nireilly@iu.edu, cartwolf@iu.edu
criekki@iu.edu, jonvance@iu.edu
peschulz@iu.edu, yijwei@iu.edu
alexschu@iu.edu, zhaozhe@iu.edu
schwani@iu.edu, jwescott@iu.edu
shethsu@iu.edu, wallachl@iu.edu
edshipp@iu.edu, wuyul@iu.edu
marystre@iu.edu, swa5@iu.edu
davthorn@iu.edu, dy11@iu.edu
sjvaleo@iu.edu, dannwint@iu.edu
jvalleci@iu.edu, sjxavier@iu.edu