C200 Programming Assignment №7
Classes, Files

Professor M.M. Dalkilic

Computer Science

School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

April 8, 2022

Introduction

The HW is due **Friday, April, 15 10:59 PM EST**. Please start **early** on this HW and submit before the deadline. Note that you get slightly more time on this HW till 10:59 on Friday.

Problem 1: Fraction Class

For this problem, you'll be given *some* of the methods for a fraction class. You are responsible for building the required functionality. We define the Fraction class as an ordered pair of positive integers (x, y). The first of the pair is called the numerator and the second the denominator. When we create an instance, the fraction is **reduced**, *i.e.*, common factors are removed. For example,

$$(2 \times 3 \times 5, 2 \times 3 \times 7) \quad \rightarrow \quad (5, 7) \tag{1}$$

We have only two operations: addition + and multiplication *; one predicate ==:

$$(x, y) + (a, b) \quad \rightarrow \quad (xb + ay, by) \tag{2}$$

$$(1, 2) + (4, 5) \quad \rightarrow \quad (5 + 8, 10) \tag{3}$$

$$(x, y) * (a, b) \quad \rightarrow \quad (xa, yb) \tag{4}$$

$$(1, 2) * (4, 5) \quad \rightarrow \quad (4, 10) \rightarrow (2(2), 2(5)) \rightarrow (2, 5) \tag{5}$$

$$(x, y) == (a, b) \quad \rightarrow \quad \begin{cases} \text{True} & x = a \text{ and } y = b \\ \text{False} & o.w. \end{cases} \tag{6}$$

When displaying an instance (x, y) we have:

$$\text{print}(\text{Fraction}(x, y)) \quad \rightarrow \quad \begin{cases} \text{Frac}(x/y) & x < y \\ \text{Frac}(a, b/c) & o.w. \end{cases} \tag{7}$$

where a is the whole number of times y divides x and b is the remainder. If $x\%y = 0$, then $b = 0$ and $c = 1$. We provide here some behavior you can inspect.

behavior Fraction class

```python
x = 2*3*5
y = 2*3*7
a = Fraction(x,y)
print(a)
b = Fraction(1,2)
c = Fraction(4,5)
d = b + c
e = b * c
print(f"{b} + {c} = {d}")
print(f"{b} * {c} = {e}")
print(Fraction(6,2))
zz = Fraction(2,4)
print(zz,b)
print(zz == b)
print(b + b == b)
```

has output:

```
frac(5/7)
frac(1/2) + frac(4/5) = frac(1,3/10)
frac(1/2) * frac(4/5) = frac(2/5)
frac(3,0/1)
frac(1/2) frac(1/2)
True
False
```

For your perusal here are the core components you will start with. Note that we have given the basic skleton as you can see below but you will need to complete the implementation. Hint: You may probably want to add some ‘get‘ (you can name it appropriately) function within this class to get the value of a fraction but that’s upto you, how you want to do that.

Fraction class

```python
class Fraction:

    def __init__(self,numerator,denominator):
       pass

    def __add__(self,other):
       pass

    def __mul__(self,other):
        pass

    def __repr__(self):
        pass

    def __eq__(self,other):
        pass
```

Deliverables Problem 1

- Complete the Fraction class
- In class we looked at Euclid's Algorithm for greatest common divisor `https://en.wikipedia.org/wiki/Euclidean_algorithm`. This is a useful algorithm to reduce the fractions, but you're free to do it as you'd like.
- While we are giving you the core methods, likely **you'll need to add others**. Remember, when calling a method within the class you write self.method(args).
- You must add get method(s) to return variables–it's up to you how you want do it.

Problem 2: Caeser Cipher

Please visit `https://en.wikipedia.org/wiki/Caesar_cipher`. You will be writing code that encrpyts and decrypts using this method. Specifically, you'll be writing functions:

$$\begin{aligned} E_n(x) &= (x+n)\ mod\ 27 \\ D_n(x) &= (x-n)\ mod\ 27 \end{aligned}$$

On the Wiki page, the modulus (%) is 26, but we're using 27–why? We are adding an extra symbol { for space. Please visit `https://en.wikipedia.org/wiki/ASCII`. If you look at the printable ASCII characters, you'll notice that (hex value 7B) follows z. Thus we can easily extend our cypher to include this symbol for space. Let's see how.

```
sentence = "this is a secret message about the class"
_sentence = sentence.replace(" ", "{")
print(_sentence)
es = ""
for i in _sentence:
    es += encrypt(i, 5)
print(es)

ds = ""
for i in es:
    ds += decrypt(i, 5)

o_sentence = ds.replace("{", " ")
print(o_sentence)
```

has output

```
1 this{is{a{secret{message{about{the{class
2 ymnxenxefexjhwjyerjxxfljefgtzyeymjehqfxx
3 this is a secret message about the class
```

In this cypher we are shifting by five. Look at the first letter ‘t’. Here is the shift in Python:

```
>>> ord('t')
116
>>> chr(ord('t') + 5)
'y'
>>> chr(ord('h') + 5)
'm'
```

Line one is our original sentence with { replacing space. Line two is the encrypted sentence. Line 3 the decrypted sentence. The shift is five, so we replace ‘t’ with ‘y’ and ‘h’ with ‘m’. What about “? It starts at the beginning at ‘a’ and returns ‘e’ which is five spaces. You are free to use chr and ord in Python or you can make a dictionary. You have complete control on how this is implemented.

What the the arguments to encrypt and decrypt? $E_n(x)$ and $D_n(x)$ take two parameters each. The letter and the amount of shift. Observe we retain the shift for all calls to both encrypt and decrypt.

Deliverables Problem 2

- Complete encrypt and decrypt. We use { to encode space
- You **can** use replace() or any string methods that Python provides
- Think of data structures that will make the problem easier

Problem 3: Central Dogma

The central dogma in biology is that DNA $\rightarrow$ RNA $\rightarrow$ protein. Please visit `https://en.wikipedia.org/wiki/Central_dogma_of_molecular_biology`. In this problem you will read in a two files: one that has how to translate three bases of DNA (codon) to an amino acid and one that has DNA.

Here is some basic helpful information to get the context (the three column table below). The first column is the name of the amino acid. The second column is the one letter initial. For the Stop_codons, we use a dash. The remaining columns are what three letters of DNA are used to make the amino acid. The amino acid Arginine has an abbreviation R. There are six codon (three bases of DNA) that code for Arginine: CGT, CGC, CGA, CGG, AGA, AGG.

```
Isoleucine, I, ATT, ATC, ATA
```

```
Leucine, L, CTT, CTC, CTA, CTG, TTA, TTG
Valine, V, GTT, GTC, GTA, GTG
Phenylalanine, F, TTT, TTC
Methionine, M, ATG
CYSteine, C, TGT, TGC
Alanine, A, GCT, GCC, GCA, GCG
Glycine, G, GGT, GGC, GGA, GGG
Proline, P, CCT, CCC, CCA, CCG
Threonine, T, ACT, ACC, ACA, ACG
Serine, S, TCT, TCC, TCA, TCG, AGT, AGC
Tyrosine, Y, TAT, TAC
Tryptophan, W, TGG
Glutamine, Q, CAA, CAG
Asparagine, N, AAT, AAC
Histidine, H, CAT, CAC
Glutamic_acid, E, GAA, GAG
AsparTic acid, D, GAT, GAC
Lysine, K, AAA, AAG
Arginine, R, CGT, CGC, CGA, CGG, AGA, AGG
Stop_codons, -, TAA, TAG, TGA
```

A FASTA file has two parts: a header (information about the sequence) and the sequence itself. Here's the one you'll be using:

```
>HSGLTH1 Human theta 1-globin gene
CCACTGCACTCACCGCACCCGGCCAATTTTTGTGTT
TTTAGTAGAGACTAAATACCATATAGTGAACACCTA
AGACGGGGGGCCTTGGATCCAGGGCGATTCAGAGG
GCCCCGGTCGGAGCTGTCGGAGATTGAGCGCGCGC
GGTCCCGGGATCTCCGACGAGGCCCTGGACCCCCG
GGCGGCGAAGCTGCGGCGCGGCGCCCCCTGGAGGC
CGCGGGACCCCTGGCCGGTCCGCGCAGGCGCAGCG
GGGTCGCAGGGCGCGGCGGGTTCCAGCGCGGGGAT
GGCGCTGTCCGCGGAGGACCGGGCGCTGGTGCGCG
CCCTGTGGAAGAAGCTGGGCAGCAACGTCGGCGTCT
ACACGACAGAGGCCCTGGAAAGGTGCGGCAGGCTG
GGCGCCCCCGCCCCCAGGGGCCCTCCCTCCCCAAG
CCCCCCGGACGCGCCTCACCCACGTTCCTCTCGCAG
GACCTTCCTGGCTTTCCCCGCCACGAAGACCTACTT
CTCCCACCTGGACCTGAGCCCCGGCTCCTCACAAGT
CAGAGCCCACGGCCAGAAGGTGGCGGACGCGCTGA
GCCTCGCCGTGGAGCGCCTGGACGACCTACCCCAC
GCGCTGTCCGCGCTGAGCCACCTGCACGCGTGCCA
```

```
GCTGCGAGTGGACCCGGCCAGCTTCCAGGTGAGCG
GCTGCCGTGCTGGGCCCCTGTCCCCGGGAGGGCCC
CGGCGGGGTGGGTGCGGGGGGCGTGCGGGGCGGG
TGCAGGCGAGTGAGCCTTGAGCGCTCGCCGCAGCT
CCTGGGCCACTGCCTGCTGGTAACCCTCGCCCGGCA
CTACCCCGGAGACTTCAGCCCCGCGCTGCAGGCGTC
GCTGGACAAGTTCCTGAGCCACGTTATCTCGGCGCT
GGTTTCCGAGTACCGCTGAACTGTGGGTGGGTGGCC
GCGGGATCCCCAGGCGACCTTCCCCGTGTTTGAGTA
AAGCCTCTCCCAGGAGCAGCCTTCTTGCCGTGCTCT
CTCGAGGTCAGGACGCGAGAGGAAGGCGC
```

You can read about this gene here: `https://pubmed.ncbi.nlm.nih.gov/3422341/`. The first line describes the sequence providing the name and other attributes. The remaining lines are the DNA sequence (ignore all whitespace).

Translating DNA into a protein

To convert from DNA to protein, we use a sequence of codons.

Let's look at the first twelve bases: CCACTGCACTCA. Every three bases uniquely determine an amino acid.

1. Start with the first codon CCA, CCACTGCACTCA.
2. Looking at the first file we see: Proline, P, CCT, CCC, CCA, CCG. This means we can rewrite CCA as P.
3. Looking at the next codon CTG, CCACTGCACTCA
4. We find it matches Leucine, L, CTT, CTC, CTA, CTG, TTA, TTG. So our protein is PL.
5. The next three are CAC CCACTGCACTCA.
6. The table has Histidine, H, CAT, CAC. We extend our string to PLH.
7. The final three are TCA. CCACTGCACTCA
8. This matches Serine, S, TCT, TCC, TCA, TCG, AGT, AGC.
9. The protein is PLHS.

If you are at the end and only have two bases, you cannot match, so you ignore them. Suppose we had CCAC. We know CCA is P. Then we only have C left. We ignore it.

In this problem, you'll read in the first table (provided as a file amino_acids.txt) and create a dictionary whose entries are:

$$aa\_d \quad = \quad \{(c_0, c_1, \ldots, c_n) : [name, letter], \ldots\}$$

where c_i is a three letter codon, $name$ is the full name of the amino acid, and $letter$ is the single letter for the amino acid. Your task is to take the DNA (from file DNA.txt) and produce a string of single letters that reflect the encoding.

The function get_amino_acid takes a path and file name and returns a dictionary. This is global, since all translations use the same code. The function get_DNA takes a path and file name and returns a list [header, DNA] (FASTA data structure) where header is the first line of the file and DNA is a string composed of all T,C,G,A characters (ignoring any whitespace). The function translate takes a FASTA data structure (the one you got from get_DNA) and returns a string that is the translation using the dictionary. In this problem we have (awkwardly) assigned the variable actual the string that is the correct translation. We can simply print to see whether our translation is the same as actual.

This code creates the dictionary and FASTA file (as a list), translates, and validates:

```
print("Dictionary")
print(aa_d)
print("FASTA file")
print(DNA_d)
print("Translations match:", str(protein == actual))
```

has output:

```
Dictionary
{('ATT', 'ATC', 'ATA'): ['Isoleucine', 'I'],
 ('CTT', 'CTC', 'CTA', 'CTG', 'TTA', 'TTG'): ['Leucine', 'L'],
 ('GTT', 'GTC', 'GTA', 'GTG'): ['Valine', 'V'],
 ('TTT', 'TTC'): ['Phenylalanine', 'F'],
 ('ATG',): ['Methionine', 'M'],
 ('TGT', 'TGC'): ['CYSteine', 'C'],
 ('GCT', 'GCC', 'GCA', 'GCG'): ['Alanine', 'A'],
 ('GGT', 'GGC', 'GGA', 'GGG'): ['Glycine', 'G'],
 ('CCT', 'CCC', 'CCA', 'CCG'): ['Proline', 'P'],
 ('ACT', 'ACC', 'ACA', 'ACG'): ['Threonine', 'T'],
 ('TCT', 'TCC', 'TCA', 'TCG', 'AGT', 'AGC'): ['Serine', 'S'],
 ('TAT', 'TAC'): ['Tyrosine', 'Y'],
 ('TGG',): ['Tryptophan', 'W'],
 ('CAA', 'CAG'): ['Glutamine', 'Q'],
 ('AAT', 'AAC'): ['Asparagine', 'N'],
 ('CAT', 'CAC'): ['Histidine', 'H'],
 ('GAA', 'GAG'): ['Glutamic_acid', 'E'],
 ('GAT', 'GAC'): ['AsparTic acid', 'D'],
 ('AAA', 'AAG'): ['Lysine', 'K'],
 ('CGT', 'CGC', 'CGA', 'CGG', 'AGA', 'AGG'): ['Arginine', 'R'],
 ('TAA', 'TAG', 'TGA'): ['Stop_codons', '-']}
FASTA file
['>HSGLTH1 Human theta 1-globin gene',
'CCACTGCACTCACCGCACCCGGCCAATTTT
TGTGTTTTTAGTAGAGACTAAATACCATATA
GTGAACACCTAAGACGGGGGGCCTTGGATC
CAGGGCGATTCAGAGGGCCCCGGTCGGAGC
TGTCGGAGATTGAGCGCGCGCGGTCCCGGG
ATCTCCGACGAGGCCCTGGACCCCCGGGCG
GCGAAGCTGCGGCGCGGCGCCCCCTGGAGG
CCGCGGGACCCCTGGCCGGTCCGCGCAGGC
GCAGCGGGGTCGCAGGGCGCGGCGGGTTCC
AGCGCGGGGATGGCGCTGTCCGCGGAGGAC
CGGGCGCTGGTGCGCGCCCTGTGGAAGAAG
CTGGGCAGCAACGTCGGCGTCTACACGACA
GAGGCCCTGGAAAGGTGCGGCAGGCTGGGC
GCCCCCGCCCCCAGGGGCCCTCCCTCCCCA
AGCCCCCCGGACGCGCCTCACCCACGTTCC
TCTCGCAGGACCTTCCTGGCTTTCCCCGCC
ACGAAGACCTACTTCTCCCACCTGGACCTG
AGCCCCGGCTCCTCACAAGTCAGAGCCCAC
GGCCAGAAGGTGGCGGACGCGCTGAGCCTC
GCCGTGGAGCGCCTGGACGACCTACCCCACG
CGCTGTCCGCGCTGAGCCACCTGCACGCGTG
CCAGCTGCGAGTGGACCCGGCCAGCTTCCAG
GTGAGCGGCTGCCGTGCTGGGCCCCTGTCCCC
GGGAGGGCCCCGGCGGGGTGGGTGCGGGGGG
CGTGCGGGGCGGGTGCAGGCGAGTGAGCCTTG
AGCGCTCGCCGCAGCTCCTGGGCCACTGCCTGC
TGGTAACCCTCGCCCGGCACTACCCCGGAGACT
TCAGCCCCGCGCTGCAGGCGTCGCTGGACAAGT
TCCTGAGCCACGTTATCTCGGCGCTGGTTTCCGA
GTACCGCTGAACTGTGGGTGGGTGGCCGCGGGA
TCCCCAGGCGACCTTCCCCGTGTTTGAGTAAAGC
CTCTCCCAGGAGCAGCCTTCTTGCCGTGCTCTCT
CGAGGTCAGGACGCGAGAGGAAGGCGC']
Translations match: True
```

Deliverables Problem 3

- We have edited the print output to make it easy to view.
- Complete the functions. Hint: You are free to add more helper functions that you think can help you to organize your code better but the main functions that are provided in the starter code must work as expected.
- You are only allowed to use replace for space and "{". You cannot use it otherwise. This might seem strange, but unfortunately, since some of the abbreviations are C,G,T,A, this approach will become difficult.

Problem 4: Univariate Function Class

In this problem we will build a class that allows us to capture a single variable function of x, evaluation of the function for a given input (i.e. the value of x), the derivative at an input, and integral over an interval. The problem does not require or even need calculus, since we're able to describe the ADT simply. The class constructor takes a Python expression of x as a string:

```
f0 = Function("1/x")
f1 = Function("x**2 - x")
f2 = Function("x**2")
```

And builds a λ function that can be applied to a well-defined input (i.e. value of x) calling the point method:

```
f0 = Function("1/x")
f1 = Function("x**2 - x")
f2 = Function("x**2")

print(f0.point(10))
print(f1.point(2))
print(f2.point(3))
```

with output

```
0.1
2
9
```

We find that a first derivative (when it exists) of f(x) can be approximated by:

$$\text{f.derivative\_at\_point(x)} = \text{(self.point(x + h) - self.point(x - h))/(2*h)} \tag{8}$$

where $h = 0.000005$.

```
print(f0.derivative_at_point(10))
print(f1.derivative_at_point(2))
print(f2.derivative_at_point(3))
```

has output

```
-0.010000000000287557
2.999999999908631
5.999999999772853
```

For students who've had calculus we can do it by hand verifying the computed answers are close. Here we are showing you how traditioanlly derivatives are done but for the HW, you should use the approximation that we have given above. The $maths$ below helps us to get an intuition behind the approximation but is not needed for the HW.

$$\begin{aligned} f(x) &= x^{-1} && (9) \\ f'(x) &= -1x^{-2} && (10) \\ f'(10) &= -1/10^2 && (11) \end{aligned}$$

$$\begin{aligned} f(x) &= x^2 - x && (12) \\ f'(x) &= 2x - 1 && (13) \\ f'(2) &= 2(2) - 1 && (14) \end{aligned}$$

$$\begin{aligned} f(x) &= x^2 && (15) \\ f'(x) &= 2x && (16) \\ f'(3) &= 2(3) && (17) \end{aligned}$$

The integral over some interval [a,b] can be appoximated by Simpson's 1/3 rule for a function f(x):

$$\begin{aligned} \text{h} &= (b-a)/4 && (18) \\ \text{d} &= \text{[a, a + h, a + 2*h, a + 3*h, a + 4*h]} && (19) \\ \text{f.integral(a,b)} &= \text{(h/3)*(f.point(d[0]) + f.point(d[4]) +} && (20) \\ & \quad \text{4*(f.point(d[1]) + f.point(d[3])) +} && (21) \\ & \quad \text{2*f.point(d[2]))} && (22) \end{aligned}$$

For example, if f(x) is 1/x using [1,2] then:

$$\begin{aligned} \text{h} &= (2-1)/4 = 0.25 && (23) \\ \text{d} &= [1.00, 1.25, 1.50, 1.75, 2.00] && (24) \\ &= (0.25/3) * [1 + 0.5 + 4 * (0.8 + 0.5714) + 2 * (0.6667)] = 0.6933 && (25) \end{aligned}$$

We can do this by hand too:

$$\begin{aligned} \int_1^2 (1/x)\,dx &= \ln(2) - \ln(1) = 0.6931471805599453 && (26) \\ \int_1^4 (x^2 - x)\,dx &= (x^3/3 - x^2/2)(4) - (x^3/3 - x^2/2)(1) && (27) \\ &= 13.33333333332 && (28) \\ \int_0^3 (x^2)\,dx &= (x^3/3)(3) - (x^3/3)(0) = 9 && (29) \end{aligned}$$

When displaying an instance simply display f(x) = body. Here is the behavior in *toto*:

```python
f0 = Function("1/x")
f1 = Function("x**2 - x")
f2 = Function("x**2")

print(f0.point(10))
print(f1.point(2))
print(f2.point(3))

print(f0.derivative_at_point(10))
print(f1.derivative_at_point(2))
print(f2.derivative_at_point(3))

print(f0.integral(1,2))
print(f1.integral(1,4))
print(f2.integral(0,3))
```

has output:

```
f(x) = 1/x
f(x) = x**2 - x
f(x) = x**2
0.1
2
9
-0.010000000000287557
2.999999999908631
5.999999999772853
0.6932539682539682
13.5
9.0
```

Deliverables Problem 4

- Complete the class
- Recall that we can use Python's eval(expression) to turn a string into a working Python expression. For example,

```python
>>> body = "3*x - 10 + (x**2)"
>>> eval("lambda x: " + body)(3)
8
```

- You cannot use any math or math-related modules for this class.

Pairs

Here are your partners for homework.

mabdayem@iu.edu, yjan@iu.edu
ahnabrah@iu.edu, dylomall@iu.edu
adamsjf@iu.edu, akaushal@iu.edu
dadeyeye@iu.edu, srpothir@iu.edu
cmaguila@iu.edu, kjwalapu@iu.edu
ahmedrr@iu.edu, ereilar@iu.edu
malshama@iu.edu, wlegear@iu.edu
olalbert@iu.edu, ssalama@iu.edu
nalemanm@iu.edu, bbacso@iu.edu
faysalza@iu.edu, gkarnuta@iu.edu
rnameen@iu.edu, chsand@iu.edu
svamin@iu.edu, fkanmogn@iu.edu
jaygul@iu.edu, njindra@iu.edu
rbajaj@iu.edu, anniye@iu.edu
cbalbuen@iu.edu, jpenrigh@iu.edu
ikbanist@iu.edu, srimmadi@iu.edu
zsbanks@iu.edu, ggivan@iu.edu
mbarrant@iu.edu, yuljiao@iu.edu
tymbarre@iu.edu, gillenj@iu.edu
dcblakle@iu.edu, lzinn@iu.edu
timbogun@iu.edu, raia@iu.edu
mlboukal@iu.edu, lburrola@iu.edu
gabradle@iu.edu, yudsingh@iu.edu
logbrads@iu.edu, jleverty@iu.edu
kbburnet@iu.edu, apapaioa@iu.edu
cbylciw@iu.edu, jhlazar@iu.edu
aidcarli@iu.edu, yangyuc@iu.edu
dcaspers@iu.edu, apoellab@iu.edu
mathchen@iu.edu, rosavy@iu.edu
joecool@iu.edu, asaokho@iu.edu
ccoriag@iu.edu, pheile@iu.edu
gcruzcor@iu.edu, kamdelmo@iu.edu
jacuau@iu.edu, fraustom@iu.edu
ddahodu@iu.edu, owenaj@iu.edu
rpdeady@iu.edu, mppan@iu.edu
cadelaga@iu.edu, sprabhak@iu.edu
emdelph@iu.edu, tclady@iu.edu
edepke@iu.edu, mdtanner@iu.edu

shrdesai@iu.edu, divpatel@iu.edu
shadoshi@iu.edu, gaoxinl@iu.edu
eeconomo@iu.edu, astrouf@iu.edu
jaespin@iu.edu, ansiva@iu.edu
mfanous@iu.edu, johnslia@iu.edu
nfarhat@iu.edu, vramkum@iu.edu
jayfish@iu.edu, ndvanbur@iu.edu
sydfoste@iu.edu, jsm13@iu.edu
ethfrago@iu.edu, mjerrell@iu.edu
nfrische@iu.edu, cl101@iu.edu
bgloor@iu.edu, sothor@iu.edu
noahgrah@iu.edu, kevko@iu.edu
halejd@iu.edu, sasaluja@iu.edu
ehallor@iu.edu, sj110@iu.edu
ejharms@iu.edu, mvincen@iu.edu
jchobbs@iu.edu, ibnash@iu.edu
binyhu@iu.edu, lopezis@iu.edu
jthurd@iu.edu, yl181@iu.edu
silmudee@iu.edu, remarche@iu.edu
aj110@iu.edu, shevphil@iu.edu
gjarrold@iu.edu, eluthra@iu.edu
cjohanns@iu.edu, owinston@iu.edu
sskauvei@iu.edu, powelchr@iu.edu
jk130@iu.edu, rwan@iu.edu
phklein@iu.edu, perkcaan@iu.edu
jtkrug@iu.edu, linweix@iu.edu
mlumbant@iu.edu, jwrohn@iu.edu
vimadhav@iu.edu, amystaff@iu.edu
gmanisca@iu.edu, evewalsh@iu.edu
pmanolis@iu.edu, evtomak@iu.edu
mmansoo@iu.edu, rlmcdani@iu.edu
tymath@iu.edu, hnasar@iu.edu
luilmill@iu.edu, dazamora@iu.edu
sahmir@iu.edu, bolabanj@iu.edu
mooralec@iu.edu, cy30@iu.edu
egmorley@iu.edu, gtutton@iu.edu
jamoya@iu.edu, lancswar@iu.edu
daparent@iu.edu, mschauss@iu.edu
aramo@iu.edu, nps1@iu.edu
mattroac@iu.edu, ryou@iu.edu
msronan@iu.edu, jjwelp@iu.edu

eliserr@iu.edu, sowvemul@iu.edu
ashankwi@iu.edu, actoney@iu.edu
grtalley@iu.edu, camitong@iu.edu
vvictori@iu.edu, chlzhang@iu.edu
samuwagn@iu.edu, lzinn@iu.edu
hdwatter@iu.edu, cadwilco@iu.edu
gkyoung@iu.edu, weidzhen@iu.edu