# C200 Programming Assignment № 2

# Functions, Containers, Choice
# Fall 2023

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

Indiana University, Bloomington, IN, USA

September 15, 2023

In this homework, you'll write functions and use choice. The homework is a little more complex than our last one–as we learn new elements, we can tackle more difficult problems. A few include quick interactive sessions–your are strongly encouraged to do these on your own.

Please start early and push often. **As always, all the work should be with you and your partner; but *both* of you should contribute.** You must complete this before **Friday, September 22, 2023 11:00PM EST**. Please remember that

- You will *not* turn anything in on Canvas.
- You will submit your work by committing your code to your GitHub repository and submitting your code to the autograder. The autograder can be accessed at `c200.luddy.indiana.edu`, and you should login via your official IU username and password.
- You do **not manually upload files** to your repository using the GitHub website's "Upload files" tool.
- You must *push* your assignment to GitHub from within VSCode *and* submit your code to the autograder.

If your timestamp is 11:01PM or later, the homework will not be graded. For any questions, please reach out ahead of time. Programming partners are shown at the end of this PDF.

Some of these problems were taken or inspired by the excellent introductory *Applied Calculus* by Tan, 2005, *Thinking Mathematically* by Blizter (5th Ed), *The Concepts and Practice of Mathematical Finance* by Joshi (2nd Ed), and others long out of print.

## 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 1-2, 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.

## Problem 1: Choice

Assume $g$ is a real-valued function defined as:

$$g(x) = \begin{cases} x + 2 & \text{if } x \neq 0 \\ 1 & \text{if } x = 0 \end{cases} \tag{1}$$

For example,

$$g(0) = 1 \tag{2}$$

$$g(1) = 3 \tag{3}$$

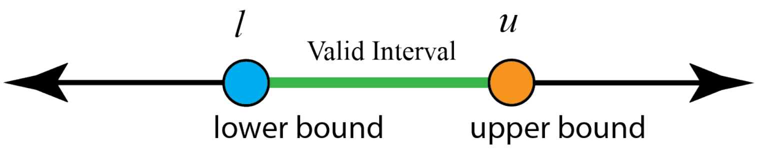
$$g(1.01) = 3.01 \tag{4}$$

**Deliverables Problem 1**

- Complete the g function

## Problem 2: Senior Citizen Health Care

Often real-world functions requires input to be validated. In this problem, the only valid years are $1977, 1978, \ldots, 1997$. The year, for example, 1976, is not valid. Python allows for a single Boolean expression for intervals. Suppose you're given an interval $[l, u]$ where $l \leq u$. The image below shows valid values (green line and filled circles) and invalid values (black lines).



Let's assume $l = 2$ and $u = 5$ and look at three values: -1.1, 3,6.

- $2 \leq -1.1 \leq 5 \rightarrow$ False
- $2 \leq 3 \leq 5 \rightarrow$ True
- $2 \leq 6 \leq 5 \rightarrow$ False

Here is a session to confirm this:

```
>>> l,u = 2,5
>>> x0, x1, x2 = -1.1, 3, 6
>>> l <= x0 <= u
False
>>> l <= x1 <= u
True
>>> l <= x2 <= u
False
>>>
```

According to a study of the out-of-pocket cost to senior citizens for health care, $f(t)$ (as percent of income), in year $t$, where $t = 0$ means 1997, is given by:

$$f(t) = \begin{cases} \frac{2}{7}t + 12 & \text{if } 0 \leq t \leq 7 \\ t + 7 & \text{if } 7 < t \leq 10 \\ \frac{3}{5}t + 11 & \text{if } 10 < t \leq 20 \end{cases} \tag{5}$$

Using a function like places burdens on the user. Let's rewrite the function so that it

- allows using an actual year *e.g.*, 1977
- return a useful error message as a string when the year is incorrect.

The new function definition is:

$$f(t) = \begin{cases} \frac{2}{7}(t - 1977) + 12 & \text{if } 1977 \leq t \leq 1984 \\ (t - 1977) + 7 & \text{if } 1984 < t \leq 1987 \\ \frac{3}{5}(t - 1977) + 11 & \text{if } 1987 < t \leq 1997 \\ \text{“YearError: } t\text{”} & \text{otherwise} \end{cases} \tag{6}$$

As you can observe, the valid years are $1977, 1978, \ldots, 1997$. Any year outside this interval is an error. The "YearError: $t$" is an exception that is raised when the wrong year is used.

For example,

$$\begin{aligned}
f(1976) &= \text{YearError: 1976} \\
f(1977) &= 12.0 \\
f(1985) &= 15 \\
f(1988) &= 17.6 \\
f(2000) &= \text{YearError: 2000}
\end{aligned}$$

**Deliverables Problem 2**

- Complete the f function.
- We aren't concerned with whether the year is an integer; in other words, it can be a float.
- When $t \notin [1977, 1997]$, then the returned **string must be** exactly like "YearError: $year$", where $year$ is the numerical value that is invalid. There is a single space after the colon.

## Problem 3: Cost of OEM parts *vs.* non-OEM parts

The cost in \$M of OEM parts for years $t = 0, 1, 2$ is given by:

$$h_0(t) = \frac{110}{(1/2)t + 1} \tag{7}$$

The cost in \$M of non OEM parts for the same years is given by:

$$h_1(t) = 26((1/4)t^2 - 1)^2 + 52 \tag{8}$$

The function that describes the difference between the costs for $t = 0, 1, 2$ is:

$$h(t) = \begin{cases} h_0(t) - h_1(t) & t \in [0, 2] \\ \text{“YearError: } t\text{”} & t \notin [0, 2] \end{cases} \tag{9}$$

For example,

$$h(0) = \$32.00 \tag{10}$$

$$h(1) = \$6.71 \tag{11}$$

$$h(1.5) = \$5.88 \tag{12}$$

$$h(2) = \$3.00 \tag{13}$$

$$h(3) = \text{“YearError: 3”} \tag{14}$$

**Deliverables Problem 3**

- Complete $h_0, h_1, h$ functions.
- Both functions $h_0, h_1$ are implemented locally within $h$.
- Round the ouptut of $h$ to two decimal places.

## Problem 4: Quadratic Formula

The roots of an equation are values that make it zero. For example,

$$x^2 - 1 = 0 \tag{15}$$

$$(x-1)(x+1) = 0 \tag{16}$$

Then $x = 1$ or $x = -1$ makes eq. 15 zero. Let's verify this. Taking $x = 1$

$$1^2 - 1 = 1 - 1 = 0 \tag{17}$$

For a quadratic equation (input variable is a power of two), there will be two roots. We'll consider complex numbers later, but for now, we'll assume the roots exist as real numbers. You learned that for a quadratic $ax^2 + bx + c = 0$, two roots $x_1, x_2$ can be determined:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{18}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{19}$$

For $x^2 - 1$ the coefficients are $a = 1, b = 0, c = -1$. Then

$$x_1 = \frac{-0 + \sqrt{0^2 - 4(1)(-1)}}{2(1)} \tag{20}$$

$$= \frac{\sqrt{4}}{2} = \frac{2}{2} = 1 \tag{21}$$

$$x_2 = \frac{-0 - \sqrt{0^2 - 4a(1)(-1)}}{2(1)} \tag{22}$$

$$= \frac{-\sqrt{4}}{2} = \frac{-2}{2} = -1 \tag{23}$$

We can report the pair of roots as (1,-1) where the left value is the larger of the two. We will assume the three values are given as a tuple $(a, b, c)$.

$$q((a,b,c)) = (\frac{-b + \sqrt{b^2 - 4ac}}{2a}, \frac{-b - \sqrt{b^2 - 4ac}}{2a}) \tag{24}$$

For example, we'll use $x^2 - 1$, $6x^2 - x - 35$, and $x^2 - 7x - 7$ :

$$q((1, 0, -1)) = (1.0, -1.0) \tag{25}$$

$$q((6, -1, -35)) = (2.5, -2.3333333333333335) \tag{26}$$

$$q((1, -7, -7)) = (7.887482193696061, -0.8874821936960613) \tag{27}$$

**Deliverables Problem 4**

- Complete the quadratic function.
- The pair of real-valued roots *must* have the larger as the first value of the tuple.
- You can not use the cmath module for this problem or any other module.

## Problem 5: AI Grading System

AI is now all over the news–disrupting almost all industries. In this problem, you'll be writing an AI system to determine whether a human has answered a simple mathematical problem correctly. Your goal is to replace instructors. Students are given an expression for example: $5 \times 5 =$ and then write an answer. corresponding answer for example: $4$. The function determines if the answer is the correct output of the given operation or not. There are four operations: multiplication, addition, subtraction, and division. The data is in the form of a list:

$$[arg_1, op, arg_2, answer]$$

$arg_1, arg_2, answer$ are floats and $op$ is the string "*", "**", "+", "-", "/". For example, [1,"*",2,3] means $1 * 2 = 3$ and is False. On the other hand, [2,"**",3,8] means $2^3 = 8$ and is True.

The function takes the data list and returns Booeans True or False. We do not do any error checking for this problem.

$$eq([arg_1, op, arg_2, answer]) = \begin{cases} \text{True} & \text{if } arg_1\ op\ arg_2 = answer \\ \text{False} & \text{otherwise} \end{cases} \tag{28}$$

Here are some examples:

$$eq([14, \text{"/"}, 2, 7])) = \text{True} \tag{29}$$
$$eq([20, \text{"*"}, 19, 381])) = \text{False} \tag{30}$$
$$eq([20, \text{"*"}, 19, 380])) = \text{True} \tag{31}$$
$$eq([2, \text{"**"}, 3, 8])) = \text{True} \tag{32}$$
$$eq([1.1, \text{"-"}, 1, .1])) = \text{False} \tag{33}$$

What's happening for the last one (33)? Remember, we have representation issues on a computer:

```
>>> 1.1 - 1
0.10000000000000009
```

We'll learn to deal with this as we progress through the semester.

**Deliverables Problem 5**

- Complete the eq function.
- You do not have to account for the zero devision error. We won't check for divide by zero case for division operation.

## Problem 6: COVID Symptom Presentation

Research from USC suggests that some COVID-19 symptoms appear in a particular order:

A. fever

B. cough and muscle pain

C. nausea or vomitting

Let's assume we want to write a program to indicate the likelihood of someone having COVID-19 based on the order in which they present symptoms A, B, and C. How many different orders are there? There are 3! = 6 possible orders (leftmost being first): ABC, CBA, CAB, BAC, BCA, ACB. The task is to determine the likelihood of COVID-19 infection given a list of any of these symptoms. For this initial problem, we'll sort the symptoms by hand, which only works for small data sets. For larger sets, we would need a quantifiable way to sort. My reasoning is that A is the most important, B is second, and C is third, but I'd have to confirm this **computational approach** biologically. For now, let's just use the following:

| very likely | likely | somewhat likely |
|---|---|---|
| ABC, ACB | BAC,BCA | CAB,CBA |

The function covid(symptoms) is given a string symptoms ABC,ACB,...,CBA. Return the string "very likely" if the pattern as 'A' as the first symptom, "likely" for the strings that have 'B' as the first symptom, and "somewhat likely" for the rest. There are many different ways to implement this. Here is a run:

```python
print(covid('ABC'),covid('ACB'))
print(covid('BAC'),covid('BCA'))
print(covid('CAB'),covid('CBA'))
```

with output:

```
very likely very likely
likely likely
somewhat likely somewhat likely
```

**Deliverables Problem 6**

- Complete the function covid.
- For the output strings-note that there is one whitespace between "very" and "likely" and one whitespace between "somewhat" and "likely".
- The returned output should be exactly as shown i.e. no trailing white spaces.

## Problem 7: Maximum

Write a function max2d that takes two numbers and returns the larger. Using only max2d write a function max3d that takes three numbers and returns the largest.

$$\begin{aligned} max2d(x,y) &= \begin{cases} x & \text{if } x > y \\ y & \text{otherwise} \end{cases} && (34) \\ max3d(x,y,z) &= max2d(x, max2d(y,z)) && (35) \end{aligned}$$

For example,

$$\begin{aligned} max3d(1,2,3) &= 3 && (36) \\ max3d(1,3,2) &= 3 && (37) \\ max3d(3,2,1) &= 3 && (38) \end{aligned}$$

**Deliverables Problem 7**

- Complete both max2d, max3d functions.
- You cannot use Python's $max()$ function.
- You can only use if, if-else or if-elif-else statements.
- $max3d$ can only use $max2d$.
- Create some of your own tests to better this problem.

Interestingly, you can use arithmetic and Boolean values for max2d (you cannot use it for this homework)

```python
def m(x,y):
    return (x > y)*x + (x <= y)*y

print(m(1,2))
print(m(2,1))
```

Has output:

```
2
2
```

## Problem 8: Voting

To complete this problem, we'll remind you of the len() function for containers and strings, but also sum(). The len() function takes a list, tuple, or string and returns the size. If the containers are empty, then zero is returned. The sum() function (which we'll implement when we learn about looping), takes either a list or tuple, and gives the sum when the members are either integers, floats, or complex numbers. We'll not consider complex numbers for this problem. For example:

```
>>> x = [1,0,1]
>>> sum(x)
2
>>> sum(x + x)
4
>>> sum((3,2,4))
9
>>> sum([1,0,1,1,0])
3
```

You'll implement the function decision(data) that takes a list [name0, name1, votes] where name0, name1 are strings for the two candidates and votes is a list of zeros and ones. A zero is a vote for name0; a 1 is a vote for name1. The function will return a tuple: (name, winning number of votes, total votes cast). Here is a run of three examples:

```
data0 = ['B','Z',[0,1,1,0,1,0,0]]
print(decision(data0))
data1 = ['B', 'Z',[1,0,1]]
print(decision(data1))
data2 = ['B', 'Z',[1,0,1,0,1,1,0,0]]
print(decision(data2))
```

produces

```
('B', 4, 7)
('Z', 2, 3)
('tie', 4, 4)
```

The first data, data0, has ‘B’ receiving 4 votes (zeros) and ‘Z" receiving 3 votes (ones); therefore, ’B’ wins with 4 votes out of 7. Similarly for the other two.

Deliverables Problem 8

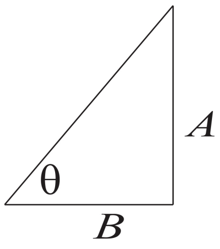
- Complete the decision(data) function. The function takes in a list of three elements: [name0, name1, votes]. The votes variable is a list of 0,1. =You can assume it is *never* empty.
- You should use len() sum() but no other built-in functions.

## Problem 9: Tan Chat

While we've seen None used in errors, its Boolean value is useful:

```
>>> not None
True
>>> bool(None)
False
>>> None and True
>>> bool(None and True)
False
>>>
```

Remember, when not coerced into a type, None doesn't return anything. This problem asks you to implement a "smart" problem solver for triangles.



To remind you, in a right triangle:

$$\tan\theta = \frac{A}{B} \tag{39}$$

$$\tan^{-1}(\tan\theta) = tan^{-1}(\frac{A}{B}) \tag{40}$$

$$\theta = tan^{-1}(\frac{A}{B}) \tag{41}$$

The equations on lines (40-41) show the inverse tangent. The math module has both tangent (as math.tan) and inverse tangent (math.atan). These use radians instead of degrees, so you'll have to convert. The math module also has functions to convert: math.radians() takes degrees and returns radians, math.degrees() takes radians and returns degrees. To remind you:

$$\pi \text{ radians} = 180 \text{ degrees}$$

In this problem, you'll write a "smart" problem solver. The user calls the function *solve(theta,opposite, adjacent)*. When one of the variables is None, the function returns the answer for that missing value. Remember, None is considered False as a Boolean value. When all the variables have values, the function returns whether the equation is True or False. Here's a run:

```
print(solve(5,None,105600))
print(solve(None,9238.9,105600))
print(solve(5,9238.8,None))
print(solve(5,9238.8,105600))
print(solve(5,9100,105600))
```

has output:

```
9238.802868337576
5.000052300754411
105599.96721475148
True
False
```

Let's see the math. For the first call, we have this where $\theta = 5^o, opposite = None, adjacent = 105600$:

$$\begin{aligned} \tan(5^o) &= \frac{None}{105600} && (42) \\ \tan(5^o)1056000 &\rightarrow 9238.802868337576 && (43) \end{aligned}$$

The function should return $\approx$ 9238.802868337576. Here's the Python in an interactive session to confirm this:

```
>>> import math
>>> math.tan(math.radians(5))*105600
9238.802868337576
```

Here's another example when $\theta = None, opposite = 9238.9, adjacent = 105600$

$$\begin{aligned} \tan(None) &= \frac{opposite}{adjacent} && (44) \\ None &= \tan^{-1}(\frac{9238.9}{105600}) \rightarrow 5.0000523007 && (45) \end{aligned}$$

Here's the Python to confirm this:

```
>>> math.degrees(math.atan(9238.8/105600))
4.999998455537281
>>> math.degrees(math.atan(9238.9/105600))
5.000052300754411
```

Observe that one decimal value difference affects the angle slightly. The hardest challenge is to determine when the equation is True because of representation. Suppose someone has this call:

```
print(solve(5,9100,105600))
```

Here are the values according to Python:

```
>>> math.tan(math.radians(5)),9238.8/105600
(0.08748866352592401, 0.08748863636363635)
```

These **should** be considered to be equal, but they are not. We can use the math function isclose() to help us. The function takes two values and an optional abs_tol = tolerance, where tolerance is a real number value, then returns True if the values up to the tolerance value specified in the abs_tol argument are the same. Where do the two values differ?

```
>>> x,y = math.tan(math.radians(5)),(9238.8/105600)
>>> x - y
2.7162287655202455e-08
>>> math.isclose(x,y,abs_tol=0.001)
True
>>> math.isclose(x,y,abs_tol=10e-9)
False
>>> math.isclose(x,y,abs_tol=10e-8)
True
```

We can see that if the tolerance is 0.001, we're sure to affirm these values are equal.

Deliverables Problem 9

- Complete the solve function.
- *hint*: you can use the variable's value of None to determine what you solve for!.
- You must use the math module including isclose().
- Use an absolute tolerance of 0.001 when implementing the isclose() function. Please read on your own if you want to know more about the function.

## Problem 10: Purchasing a Home without PMI

We saw the formula for an annuity with interest $r$ compounded yearly $n$ over $t$ years with a payment of $P$:

$$A = \frac{P((1+\frac{r}{n})^{nt}-1)}{(\frac{r}{n})} \tag{46}$$

This formula can be used to find what your payment needs to be to finance, say, a house. When purchasing a house, if you're unable to pay at least 20% of the price, you're required to pay an additional 0.2% to 3.0% (Private Mortgage Insurance) of the original loan. PMI is a heavy burden. You want to write a program that, given a home's price $H$, allows you to determine what is the minimum down payment to avoid PMI. For this problem we will assume you want to have the down payment in two years $t = 2$ and you'll make monthly payments $n = 12$. For example, a home costing \$250,000 requires $.2(\$250,000) = \$50,000$ due at signing. Assuming you're getting 6% compounded monthly for two years, then you'll need to pay \$1,966.03 per month. Here is a run:

```
1  #problem 10
2      home_price, rate = 250000, 6/100
3      t,n = 2,12                         #years, monthly
4      payment = future(home_price,rate)
5
6      print(f"{n} payments yearly for {t} years requires ${payment}")
7
8      #confirm this achieves 50000
9      A = round(payment*((1 + rate/n)**(t*n)-1)/(rate/n),2)
10     print(A)
```

producing

```
12 payments yearly for 2 years requires $1966.03
49999.99
```

Line 8 uses Eq. 46. As not unexpected, there's a tiny roundoff problem. A penny won't make a difference.

**Deliverables Problem 10**

- To implement the function, you'll need to solve for $P$ in Eq. 46.
- For this problem, we assume $n = 12$ and $t = 2$, *i.e.*, monthly payments for two years. Therefore, the formal parameters for the problem are home price and interest rate.
- The function future(A,r) returns the payment rounded to two decimal places.

## Programming partners

liansia@iu.edu, nichojop@iu.edu
ovadeley@iu.edu, awolkind@iu.edu
tcconnol@iu.edu, krbpatel@iu.edu
clearle@iu.edu, mmarotti@iu.edu
bencalex@iu.edu, dwo@iu.edu
bencho@iu.edu, ajtse@iu.edu
dblackme@iu.edu, ryarram@iu.edu
jeonjunh@iu.edu, mehtriya@iu.edu
aberkun@iu.edu, lmadiraj@iu.edu
mwclawso@iu.edu, thnewm@iu.edu
leegain@iu.edu, audtravi@iu.edu
allencla@iu.edu, jsadiq@iu.edu
andhugo@iu.edu, leolin@iu.edu
eakanle@iu.edu, clscheum@iu.edu
egoldsto@iu.edu, emluplet@iu.edu
ek37@iu.edu, aranjit@iu.edu
enahern@iu.edu, wardjohn@iu.edu
ohostet@iu.edu, ysanghi@iu.edu
jc168@iu.edu, giomayo@iu.edu
gandhira@iu.edu, meyerli@iu.edu
bellcol@iu.edu, apathma@iu.edu
nihanas@iu.edu, cltran@iu.edu
tfleuran@iu.edu, sahaan@iu.edu
alchatz@iu.edu, justyou@iu.edu
oeichenb@iu.edu, moralemd@iu.edu
ajgrego@iu.edu, utwade@iu.edu
nharkins@iu.edu, maglowe@iu.edu
bcdutka@iu.edu, woodsky@iu.edu
marganey@iu.edu, cinivo@iu.edu
bkante@iu.edu, cmvanhov@iu.edu
migriswo@iu.edu, jlopezmo@iu.edu
jwcase@iu.edu, myswill@iu.edu
willhowa@iu.edu, fshamrin@iu.edu
jhhudgin@iu.edu, ntorpoco@iu.edu
evacoll@iu.edu, kvpriede@iu.edu
alelefeb@iu.edu, skp2@iu.edu
jwdrew@iu.edu, madymcsh@iu.edu
ejhaas@iu.edu, ansakrah@iu.edu
linjaso@iu.edu, vpolu@iu.edu

blacount@iu.edu, nmulheri@iu.edu
schinitz@iu.edu, dsummit@iu.edu
abuswell@iu.edu, joshroc@iu.edu
hh35@iu.edu, megapaul@iu.edu
caegrah@iu.edu, ap79@iu.edu
hermbrar@iu.edu, scotbray@iu.edu
ameydesh@iu.edu, thomps16@iu.edu
aakindel@iu.edu, majtorm@iu.edu
alyjone@iu.edu, blswing@iu.edu
greenpat@iu.edu, aveluru@iu.edu
coopelki@iu.edu, rorymurp@iu.edu
nbernot@iu.edu, gmpierce@iu.edu
saecohen@iu.edu, smremmer@iu.edu
fu7@iu.edu, mveltri@iu.edu
kraus@iu.edu, reedkier@iu.edu
austdeck@iu.edu, kamaharj@iu.edu
apchavis@iu.edu, samyuan@iu.edu
apbabu@iu.edu, clmcevil@iu.edu
quecox@iu.edu, cialugo@iu.edu
kjj6@iu.edu, rpoludas@iu.edu
coopjose@iu.edu, jacmanto@iu.edu
colrkram@iu.edu, pp31@iu.edu
davgourl@iu.edu, rvinzant@iu.edu
johnguen@iu.edu, jneblett@iu.edu
avulas@iu.edu, lukastef@iu.edu
amkhatri@iu.edu, aptarin@iu.edu
fkeele@iu.edu, marystre@iu.edu
ageorgio@iu.edu, vmungara@iu.edu
wilcusic@iu.edu, bcmarret@iu.edu
arklonow@iu.edu, wtubbs@iu.edu
laharden@iu.edu, jarlmint@iu.edu
tfreson@iu.edu, wattsra@iu.edu
jkielcz@iu.edu, jcn1@iu.edu
maladwa@iu.edu, hasiddiq@iu.edu
krisgupt@iu.edu, jwember@iu.edu
nfelici@iu.edu, ap52@iu.edu
bjdahl@iu.edu, arirowe@iu.edu, sezinnkr@iu.edu
twfine@iu.edu, jaslnu@iu.edu
cpkerns@iu.edu, myeralli@iu.edu
alscarr@iu.edu, jtsuter@iu.edu
cgkabedi@iu.edu, reddyrr@iu.edu

ethickma@iu.edu, avraya@iu.edu
diebarro@iu.edu, asaokho@iu.edu
jakchap@iu.edu, masmatth@iu.edu
hk120@iu.edu, amanocha@iu.edu
nmcastan@iu.edu, mszczas@iu.edu
anrkram@iu.edu, erschaef@iu.edu
celechav@iu.edu, maklsmit@iu.edu
jdemirci@iu.edu, deturne@iu.edu
makinap@iu.edu, ksadiq@iu.edu
scbik@iu.edu, iperine@iu.edu
ddrotts@iu.edu, snyderjk@iu.edu
mkleinke@iu.edu, jwu6@iu.edu
mohiambu@iu.edu, shrrao@iu.edu
sharor@iu.edu, cnyarko@iu.edu
huhasan@iu.edu, jpochyly@iu.edu
kaihara@iu.edu, fmahamat@iu.edu
simadams@iu.edu, anemlunc@iu.edu
cuizek@iu.edu, vyeruba@iu.edu
dkkosim@iu.edu, tangtom@iu.edu
aroraarn@iu.edu, lreddell@iu.edu
jdc6@iu.edu, samrile@iu.edu
jhar@iu.edu, mz24@iu.edu
braybrya@iu.edu, aidnschi@iu.edu
ballachm@iu.edu, nmr1@iu.edu
earuland@iu.edu, mmunaf@iu.edu
escolber@iu.edu, amyawash@iu.edu
kekchoe@iu.edu, dmmullin@iu.edu
tchapell@iu.edu, wilsonnf@iu.edu
ethbrock@iu.edu, jdw14@iu.edu
agawrys@iu.edu, zshamo@iu.edu
micahand@iu.edu, asteini@iu.edu
mrcoons@iu.edu, ragmahaj@iu.edu
nokebark@iu.edu, asultano@iu.edu
deombeas@iu.edu, trenstev@iu.edu
spgerst@iu.edu, keasandl@iu.edu
zhatfie@iu.edu, gsilingh@iu.edu
mbekkem@iu.edu, rosenbbj@iu.edu
oakinsey@iu.edu, mjroelle@iu.edu
sydecook@iu.edu, emisimps@iu.edu
ryanbren@iu.edu, ijvelmur@iu.edu
milhavil@iu.edu, ir1@iu.edu

sgaladim@iu.edu, rnschroe@iu.edu
ruqchen@iu.edu, tarturnm@iu.edu
micedunn@iu.edu, jwmullis@iu.edu
masharre@iu.edu, nniranj@iu.edu
rl29@iu.edu, gszopin@iu.edu
ajeeju@iu.edu, gavsteve@iu.edu
kaneai@iu.edu, emgward@iu.edu
mdonato@iu.edu, brayrump@iu.edu
sakalwa@iu.edu, patelsak@iu.edu
ag69@iu.edu, tpandey@iu.edu
edfran@iu.edu, lmeldgin@iu.edu
maxklei@iu.edu, wilsori@iu.edu
dja1@iu.edu, ltmckinn@iu.edu
mbrockey@iu.edu, pw18@iu.edu
nfarhat@iu.edu, bmpool@iu.edu
mbeigie@iu.edu, pravulap@iu.edu
sg40@iu.edu, dukthang@iu.edu
matgarey@iu.edu, patedev@iu.edu
stkimani@iu.edu, wtatoole@iu.edu
garcied@iu.edu, surapapp@iu.edu
hawkjod@iu.edu, rwan@iu.edu
zguising@iu.edu, msorsor@iu.edu
fkanmogn@iu.edu, gs29@iu.edu
daminteh@iu.edu, anassour@iu.edu
maudomin@iu.edu, muyusuf@iu.edu
ckdiallo@iu.edu, patekek@iu.edu
agrevel@iu.edu, tolatinw@iu.edu
phjhess@iu.edu, ao9@iu.edu
vkommar@iu.edu, liwitte@iu.edu
flynncj@iu.edu, cjwaller@iu.edu
skunduru@iu.edu, lpelaez@iu.edu
browdeon@iu.edu, antando@iu.edu
khannni@iu.edu, gmichna@iu.edu
leokurtz@iu.edu, lqadan@iu.edu
brhint@iu.edu, bdyiga@iu.edu
dce@iu.edu, mnimmala@iu.edu
tychid@iu.edu, aledminc@iu.edu
grafe@iu.edu, anajmal@iu.edu
spdamani@iu.edu, wtrucker@iu.edu
gcopus@iu.edu, tt13@iu.edu
jabbarke@iu.edu, ryminer@iu.edu

amkurz@iu.edu, schwajaw@iu.edu
harmonnc@iu.edu, benprohm@iu.edu
mkames@iu.edu, emmmccle@iu.edu
lflenoy@iu.edu, ntuhl@iu.edu
kapgupta@iu.edu, singkani@iu.edu
ridbhan@iu.edu, voram@iu.edu
mrfehr@iu.edu, wwtang@iu.edu
adwadash@iu.edu, btasa@iu.edu
nabussel@iu.edu, crmoll@iu.edu
joehawl@iu.edu, ilynam@iu.edu
brownset@iu.edu, mzagotta@iu.edu
zacbutle@iu.edu, orrostew@iu.edu
loggreen@iu.edu, evataylo@iu.edu
lpfritsc@iu.edu, nrizvi@iu.edu
aleliffe@iu.edu, adsize@iu.edu
wanjiang@iu.edu, jactrayl@iu.edu
jacklapp@iu.edu, cmarcuka@iu.edu
sfuneno@iu.edu, etprince@iu.edu
criscruz@iu.edu, abiparri@iu.edu
spgreenf@iu.edu, aptheria@iu.edu
jdgonzal@iu.edu, phamvinh@iu.edu
mdiazrey@iu.edu, apavlako@iu.edu
rcaswel@iu.edu, jlzhao@iu.edu
anlego@iu.edu, sasayini@iu.edu
kyhigg@iu.edu, owysmit@iu.edu
abellah@iu.edu, asidda@iu.edu
mdoxsee@iu.edu, chrimanu@iu.edu
seangarc@iu.edu, pateishi@iu.edu
cannan@iu.edu, rt11@iu.edu
elybrewe@iu.edu, lvansyck@iu.edu
howamatt@iu.edu, gepearcy@iu.edu
skatiyar@iu.edu, isaramir@iu.edu
jtbland@iu.edu, jwetherb@iu.edu
cfampo@iu.edu, perezand@iu.edu
jacobben@iu.edu, rafir@iu.edu
lcoveney@iu.edu, aditpate@iu.edu
ahavlin@iu.edu, pricemo@iu.edu
aaragga@iu.edu, nsatti@iu.edu
josespos@iu.edu, cstancom@iu.edu
lawmat@iu.edu, epautsch@iu.edu
arnadutt@iu.edu, sahimann@iu.edu

adiyer@iu.edu, savebhat@iu.edu
marcchao@iu.edu, mimimurp@iu.edu
aketcha@iu.edu, dernguye@iu.edu
gracarmi@iu.edu, impofujr@iu.edu
keswar@iu.edu, wlyzun@iu.edu
gmhowell@iu.edu, awsaunde@iu.edu
swconley@iu.edu, ism1@iu.edu
kdembla@iu.edu, gavilleg@iu.edu
daxbills@iu.edu, coenthom@iu.edu
howardbw@iu.edu, nlippman@iu.edu
nolakim@iu.edu, dyashwar@iu.edu
saganna@iu.edu, ruska@iu.edu
althart@iu.edu, jnzheng@iu.edu
aaamoako@iu.edu, jomeaghe@iu.edu
delkumar@iu.edu, ammulc@iu.edu
adhuria@iu.edu, rtrammel@iu.edu
hamac@iu.edu, patel89@iu.edu
laburkle@iu.edu, annaum@iu.edu
eghough@iu.edu, qshamsid@iu.edu
wgurley@iu.edu, aayushah@iu.edu
achordi@iu.edu, violuway@iu.edu
fdonfrio@iu.edu, tzuyyen@iu.edu
efritch@iu.edu, nmata@iu.edu
bcison@iu.edu, aselki@iu.edu
pyphealy@iu.edu, vrradia@iu.edu
ceub@iu.edu, sahishah@iu.edu
ayoajayi@iu.edu, aamathew@iu.edu
jongrim@iu.edu, drsnid@iu.edu