C200 Programming Assignment № 4

Functions, Containers, Choice
Spring 2025

Dr. M.M. Dalkilic

Computer Science

School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

February 23, 2025

In this homework, you'll write functions, use choice, and loop through items. The homework is a little more complex than our last one–as we learn new elements, we can tackle more difficult problems. To help better understand container assignment, several of the functions have the initial assignments done for you. You can use other assignments, of course and safely ignore them.

As always, all the work should be with you and your partner; but *both* of you should contribute. You must complete this before **Friday, February 28 2025 5:00PM EST**. Please remember that

- You will *not* turn anything in on Canvas.
- You will be submitting on Gradescope
- Your highest score of **10 submissions** will be used for your final score.
- Any function that is defined requires a docstring, please refer to lab 03 for what a docstring should have.
- Finally, manual grading will be done, so if you receive a 50 out of 100 on the autograder, it means the lowest grade you can receive is 50. The rest will come from manual grading of the problems.

If your timestamp is 5:01PM or later, the homework will not be graded. So **do not wait until 4:59 PM** to submit. If you have questions or problems with Gradescope, please visit office hours or make a post on Inscribe ahead of time. Since you are working in pairs, your paired partner is shown on canvas.

Some of these problems were taken or inspired by the excellent introductory *Applied Calculus* by Tan, 2005.

Instructions for submitting to Gradescope

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 IDLE and watching for unexpected behavior such as, program failing with syntax error or not returning correct output.

3. Remember to always **test, test, and test** your code before submitting. The autograder is not meant to be a substitute for testing your code.

4. If the autograder does not appear under the assignment, then you can try the website, https://www.gradescope.com/.

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
- Include a docstring for the function.

Problem 2: Senior Citizen Health Care

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

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

We will contextualize this function by corresponding $t = 0$ to 1977. To do this, we can employ $t - 1977$. We'll assume $t \in [1977, 1997]$, meaning that t must be in this interval. If $t \notin [1977, 1997]$, then we return a string "error: year". The new function definition is:

$$cost(t) \quad = \quad \begin{cases} \frac{2}{7}(t - 1977) + 12 & \text{if } 1977 \leq t \leq 1984 \\ (t - 1977) + 7 & \text{if } 1984 < t \leq 1987 \\ \frac{3}{5}(t - 1977) + 11 & \text{if } 1987 < t \leq 1997 \\ \text{"error: year"} & \text{otherwise} \end{cases} \tag{6}$$

For example,

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

Deliverables Problem 2

- Complete the cost function.
- Include a docstring for the function.
- When $t \notin [1977, 1997]$, then the returned string must be exactly "error: year". There is no space between error and ":", and only 1 whitespace between ":" and "year".

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

The cost of OEM parts for year $t = 0$, year $t = 1$, and year $t = 2$ is given by:

$$oem_0(t) = \frac{110}{(1/2)t + 1} \tag{7}$$

The cost of non OEM parts for the same years is given by:

$$oem_1(t) = 26((1/4)t^2 - 1)^2 + 52 \tag{8}$$

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

$$oem(t) = oem_0(t) - oem_1(t) \tag{9}$$

For example,

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

$$oem(1) \approx \$6.71 \tag{11}$$

$$oem(2) = \$3.00 \tag{12}$$

Deliverables Problem 3

- Complete oem_0, oem_1, oem functions.
- Include a docstring for the function(s).
- Round to two decimal places for oem.

Problem 4: Quadratic Formula

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

$$x^2 - 1 = 0 \tag{13}$$
$$(x-1)(x+1) = 0 \tag{14}$$

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

$$(1)^2 - 1 = 0 \tag{15}$$
$$(1-1)(1+1) = 0 \tag{16}$$
$$\tag{17}$$

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).

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

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

$$quad((1,0,-1)) = (1.0, -1.0) \tag{25}$$
$$quad((6,-1,-35)) = (2.5, -2.3333333333333335) \tag{26}$$
$$quad((1,-7,-7)) = (7.887482193696061, -0.8874821936960613) \tag{27}$$

Deliverables Problem 4

- Complete the quadratic function.
- You can not use the cmath module for this problem or any other module.
- Include a docstring for the function.

Problem 5: AI Grading System

You are writing an AI system to help students in arithmetic. Students are given an expression for example: $5 \times 5 =$ and a 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 a string "*", "+", "-", "/". For example, [1,"*",2,3] which is $1 * 2 = 3$. This expression is False.

The function takes the data list and returns True or False. We'll assume the arguments and answer are the correct type.

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

Here are some examples:

$$expr\_eq([14, \text{"/"}, 2, 7]) \quad = \quad \text{True} \tag{29}$$

$$expr\_eq([20, \text{"*"}, 19, 381]) \quad = \quad \text{False} \tag{30}$$

$$expr\_eq([1.1, \text{"-"}, 1, .1])) \quad = \quad \text{False} \tag{31}$$

Deliverables Problem 5

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

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 vomiting

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 chance_of_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:

```
1 print(chance_of_covid('ABC'),chance_of_covid('ACB'))
2 print(chance_of_covid('BAC'),chance_of_covid('BCA'))
3 print(chance_of_covid('CAB'),chance_of_covid('CBA'))
```

with output:

```
1 very likely very likely
2 likely likely
3 somewhat likely somewhat likely
```

Deliverables Problem 6

- Complete the function chance_of_covid.
- For the output strings-note that there is one whitespace between "very" and "likely" and one whitespace between "somewhat" and "likely".
- Include a docstring for the function.

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.

$$max2d(x, y) = \begin{cases} x & \text{if } x > y \\ y & \text{otherwise} \end{cases} \tag{32}$$

$$max3d(x, y, z) = max2d(x, max2d(y, z)) \tag{33}$$

For example,

$$max3d(1, 2, 3) = 3 \tag{34}$$

$$max3d(1, 3, 2) = 3 \tag{35}$$

$$max3d(3, 2, 1) = 3 \tag{36}$$

Deliverables Problem 7

- Complete both max2d, max3d functions.
- You cannot use Python max.
- You can only use if statements.
- max3d can only use max2d.
- Include a docstring for the function.

Problem 8: Lines in 2D

Although you know it, we'll remind you about lines: Given two points $p_0 = (x_0, y_0), p_1 = (x_1, y_1)$, a line is formed by:

$$\begin{aligned} y &= mx + b \\ m &= \frac{y_0 - y_1}{x_0 - x_1} \end{aligned}$$

Using either point, we can solve for the intercept b. For a non-zero slope m, the inverse is $-m^{-1}$. Three points p_0, p_1, p_2 are colinear if they lie along the same line. You've decided to create a start-up that helps K-12 students learn about lines–in particular, 2D Euclidean space. After researching the area, you decided on three functions:

1. build_line that takes two points and a dictionary and assigns the keys "slope" and returns the dictionary containing the slope and intercept if the points define a line–remember, any two points $(x, y_0), (x, y_1)$ does not because the slope is undefined.
2. clear_line that assigns None to both the slope and intercept.
3. inverse_slope that returns the inverse of the slope: if m is the slope, the inverse is $-m^{-1}$. Since $m = 0$ does not have an inverse, if there's not a valid way to return the inverse, you return "Error: no slope".
4. colinear that takes three points p_0, p_1, p_2 and, using build_line returns True if they are colinear and False otherwise

To make things easier you've decided to use a dictionary:

```
line = {'slope':None, 'intercept':None}
```

This dictionary holds the slope and intercept, but begins with None. Here's a simple run:

```
print(line)
build_line((2,3),(8,6),line)
print(line)
clear_line(line)
print(line)
build_line((2,3),(2,5),line) #not a line (no slope)
print(line)
```

with output:

```
{'slope': None, 'intercept': None}
{'slope': 0.5, 'intercept': 2.0}
{'slope': None, 'intercept': None}
{'slope': None, 'intercept': None}
```

The function call build_line assigns to the keys "slope" and "intercept" the slope and intercept if the two points define a line; otherwise, the dictionary is not changed as shown in the last print. Let's see the inverse slope and colinear functions:

```
build_line((-3,2),(4,-1),line)
print(line)
print(inverse_slope(line))
print(colinear((-2,1),(1,7),(4,13)))
```

has output:

```
{'slope': -0.42857142857142855, 'intercept': 0.7142857142857144}
2.3333333333333335
True
```

When you send a mutable object to a function, a new local copy is **not** created: it uses the same address. Observe this code:

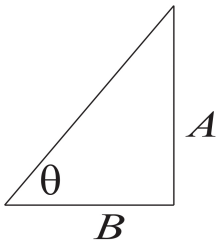
```
>>> x = ['dog',123]
>>> def f(happy):
...     happy[1] = 'who let the'
...
>>> x
['dog', 123]
>>> f(x)
>>> x
['dog', 'who let the']
```

We will change the mutable dictionary as an argument. Later in the course, we'll learn how to make a copy of the mutable object.

Deliverables Problem 8

- Complete the build_line, inverse_slope and colinear functions.
- We've completed the clear function.
- You must use the dictionary line.
- The function colinear **must** use the build_line function. Use the local dictionary line that we have provided.
- Include a docstring for the function.

Problem 9: Tan Chat



To remind you, in a right triangle:

$$\tan\theta = \frac{A}{B} \tag{37}$$

$$\tan^{-1}(\tan\theta) = tan^{-1}(\frac{A}{B}) \tag{38}$$

$$\theta = tan^{-1}(\frac{A}{B}) \tag{39}$$

The equation on line 36 shows 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:

```
1 print(solve(5,None,105600))
2 print(solve(None,9238.9,105600))
3 print(solve(5,9238.8,None))
4 print(solve(5,9238.8,105600))
5 print(solve(5,9100,105600))
```

has output:

```
1 9238.802868337576
2 5.000052300754411
3 105599.96721475148
4 True
5 False
```

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

$$\tan(5^o) = \frac{None}{105600} \tag{40}$$

$$\tan(5^o)1056000 \rightarrow 9238.802868337576 \tag{41}$$

The function should return ≈ 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 $\theta = None, opposite = 9238.9, adjacent = 105600$

$$\tan(None) = \frac{opposite}{adjacent} \tag{42}$$

$$None = \tan^{-1}(\frac{9238.9}{105600}) \rightarrow 5.0000523007 \tag{43}$$

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.
- Include a docstring for the function.

Problem 10: Toward Statistical Analysis

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

$$
\begin{aligned}
mean(lst) &= (x_0 + x_1 + \cdots + x_n)/n && (44)\\
\mu &= mean(lst) && (45)\\
var(lst) &= \frac{1}{n}((x_0 - \mu)^2 + (x_1 - \mu)^2 + \cdots + (x_n - \mu)^2) && (46)\\
std(lst) &= \sqrt{var(lst)} && (47)
\end{aligned}
$$

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

$$
\begin{aligned}
mean(lst) &= 4.67 && (48)\\
var(lst) &= 12.22 && (49)\\
std(lst) &= 3.5 && (50)
\end{aligned}
$$

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:

$$
\begin{aligned}
\mu &= (x_0 + x_1 + \cdots + x_n)/n && (51)\\
lst &= [x_0 - \mu, x_1 - \mu, \ldots x_n - \mu] && (52)\\
mean(lst) &= ((x_0 - \mu) + \cdots (x_n - \mu))/n && (53)\\
&= ((x_0 + \ldots + x_n) + n\mu)/n && (54)\\
&= \mu - \mu = 0 && (55)
\end{aligned}
$$

Using the same list we have

$$
mean(mean\_centered(lst)) = -0.0 = 0 \quad (56)
$$

Deliverables for Problem 10

- Complete the functions.
- You **cannot** use *sum*
- Round-up to 2 decimal places the ouptut of mean, var and std functions.
- Include a docstring for the function.

Problem 11: Fib & Trib

You may have heard of the fibonacci numbers, which are a series of numbers generated by the following function:

$$fibonacci(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ fibonacci(n-1) + fibonacci(n-2), & n > 1 \end{cases} \tag{57}$$

Which produces the following series of numbers for $n < 10$:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34$$

Each fibonacci number is generated by adding the previous two, other than in the case of $n = 0$ and $n = 1$.

Now consider the tribonacci numbers, which are a series of numbers generated by the following function:

$$tribonacci(n) = \begin{cases} 0, & n = 0 \\ 0, & n = 1 \\ 1, & n = 2 \\ tribonacci(n-1) + tribonacci(n-2) + tribonacci(n-3), & n > 2 \end{cases} \tag{58}$$

Which produces the following series of numbers for $n < 10$:

$$0, 0, 1, 1, 2, 4, 7, 13, 24, 44$$

To aid in the solution of this problem, recall that factorial is also defined in a self referential way:

$$fact(n) = \begin{cases} 0, & n = 0 \\ n * fact(n-1), & n > 0 \end{cases} \tag{59}$$

Yet, when writing the solution to factorial, there is no need to be self referential:

```
1 def fact(n):
2     fact_n = 1
3     if n == 0:
4         return fact_n
5     for m in range(1, n+1):
6         fact_n = fact_n * m
7     return fact_n
```

Deliverables Problem 11

- Complete the fibonacci & tribonacci functions using **for loops.**
- Include a docstring for the function.
- **Hint:** You will want to have some variables inside of your function *seeded* to the right value for the simple cases of both functions, since those require no other computation and can be returned directly. They can also be used to compute the later values of the series.