C200 Programming Assignment № 2

Functions, Containers, Choice
Spring 2022

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

Indiana University, Bloomington, IN, USA

February 4, 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. 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.

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, February 10 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. So **do not wait until 10:59 PM** to commit and push your changes. If you have questions or problems with version control, please visit office hours or make a post on Inscribe ahead of time. Since you are working in pairs, your paired partner is shown at the end of this PDF.

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

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.

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,

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

$$g(1) = 2 \tag{3}$$

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

Deliverables Problem 1

- Complete the g function

Problem 2: Senior Citizen Health Care

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 $t = 0$, is given by:

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

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:

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

For example,

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

Deliverables Problem 2

- Complete the f 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:

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

The cost 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:

$$h(t) = h_0(t) - h_1(t) \tag{9}$$

For example,

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

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

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

Deliverables Problem 3

- Complete h_0, h_1, h functions.
- Round to two decimal places.

Problem 4: Quadratic Formula

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

$$
\begin{aligned}
x^2 - 1 &= 0 && (13)\\
(x-1)(x+1) &= 0 && (14)
\end{aligned}
$$

Then $x = 1$ or $x = -1$ makes eq. 13 zero. Let's verify this. Taking $x = 1$

$$1^2 - 1 = 1 - 1 = 0 \qquad (15)$$

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:

$$
\begin{aligned}
x_1 &= \frac{-b + \sqrt{b^2 + 4ac}}{2a} && (16)\\
x_2 &= \frac{-b - \sqrt{b^2 + 4ac}}{2a} && (17)
\end{aligned}
$$

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

$$
\begin{aligned}
x_1 &= \frac{-0 + \sqrt{0^2 - 4(1)(-1)}}{2(1)} && (18)\\
&= \frac{\sqrt{4}}{2} = \frac{2}{2} = 1 && (19)\\
x_2 &= \frac{-0 - \sqrt{0^2 - 4a(1)(-1)}}{2(1)} && (20)\\
&= \frac{-\sqrt{4}}{2} = \frac{-2}{2} = -1 && (21)
\end{aligned}
$$

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).

$$q((a,b,c)) = \left(\frac{-b + \sqrt{b^2 - 4ac}}{2a}, \frac{-b - \sqrt{b^2 - 4ac}}{2a}\right) \qquad (22)$$

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

$$
\begin{aligned}
q((1, 0, -1)) &= (1.0, -1.0) && (23)\\
q((6, -1, -35)) &= (2.5, -2.3333333333333335) && (24)\\
q((1, -7, -7)) &= (7.887482193696061, -0.8874821936960613) && (25)
\end{aligned}
$$

Deliverables Problem 4

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

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.

$$eq([arg_1, op, arg_2, answer]) = \begin{cases} \text{True} & \text{if } arg_1\ op\ arg_2 = answer \\ \text{False} & \text{otherwise} \end{cases} \tag{26}$$

Here are some examples:

$$eq([14, \text{"/"}, 2, 7]) = \text{True} \tag{27}$$
$$eq([20, \text{"*"}, 19, 381]) = \text{False} \tag{28}$$
$$eq([1.1, \text{"-"}, 1, .1])) = \text{False} \tag{29}$$

We did the last one in class as an example of representation problems.

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".

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.

$$max2d(x, y) = \begin{cases} x & \text{if } x > y \\ y & \text{otherwise} \end{cases} \quad (30)$$

$$max3d(x, y, z) = max2d(x, max2d(y, z)) \quad (31)$$

For example,

$$max3d(1, 2, 3) = 3 \quad (32)$$

$$max3d(1, 3, 2) = 3 \quad (33)$$

$$max3d(3, 2, 1) = 3 \quad (34)$$

Deliverables Problem 7

- Complete both max2d, max3d functions.
- You cannot use Python max.
- You can only use if statements.
- max3d can only use max2d.

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: 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:

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

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

```python
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:

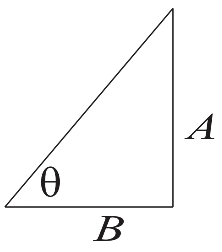
```
>>> x = ['dog',123]
>>> def f(happy):
...     happy[1] = 'who let the'
...
>>> x
['dog', 123]
>>> f(x)
>>> x
['dog', 'who let the']
```

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.

Problem 9: Tan Chat



To remind you, in a right triangle:

$$\tan\theta = \frac{A}{B} \tag{35}$$

$$\tan^{-1}(\tan\theta) = tan^{-1}(\frac{A}{B}) \tag{36}$$

$$\theta = tan^{-1}(\frac{A}{B}) \tag{37}$$

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:

```python
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 $\theta = 5^o, opposite = None, adjacent = 105600$:

$$\tan(5^o) = \frac{None}{105600} \tag{38}$$

$$\tan(5^o)1056000 \rightarrow 9238.802868337576 \tag{39}$$

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 $\theta = None, opposite = 9238.9, adjacent = 105600$

$$\tan(None) = \frac{opposite}{adjacent} \quad (40)$$

$$None = \tan^{-1}(\frac{9238.9}{105600}) \rightarrow 5.0000523007 \quad (41)$$

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.

Student Pairs

aaberma@iu.edu, lsherbst@iu.edu
actonm@iu.edu, jacosby@iu.edu, ss126@iu.edu
adagar@iu.edu, jmom@iu.edu
brakin@iu.edu, achimeba@iu.edu
sakinolu@iu.edu, jarymeln@iu.edu
moalnass@iu.edu, cmarcucc@iu.edu
anderblm@iu.edu, hnichin@iu.edu
jaybaity@iu.edu, sihamza@iu.edu
nbakken@iu.edu, mszczas@iu.edu
chnbalta@iu.edu, sfawaz@iu.edu
nokebark@iu.edu, chrinayl@iu.edu
nwbarret@iu.edu, cahilber@iu.edu
sbehman@iu.edu, ivycai@iu.edu
jadbenav@iu.edu, mgambett@iu.edu
arjbhar@iu.edu, agoldswo@iu.edu
dombish@iu.edu, cartwolf@iu.edu
gavbish@iu.edu, parisbel@iu.edu
sebisson@iu.edu, sjkallub@iu.edu
aibitner@iu.edu, chnico@iu.edu
abolad@iu.edu, sgcolett@iu.edu
hjbolus@iu.edu, grschenc@iu.edu
jacbooth@iu.edu, psaggar@iu.edu
joshbrin@iu.edu, egillig@iu.edu
neybrito@iu.edu, sijatto@iu.edu
nbulgare@iu.edu, aketcha@iu.edu
bcarl@iu.edu, hvelidi@iu.edu
ecastano@iu.edu, mdgamble@iu.edu

cheng47@iu.edu, abdufall@iu.edu
kbchiu@iu.edu, trewoo@iu.edu
hc51@iu.edu, lmamidip@iu.edu
joracobb@iu.edu, dannwint@iu.edu
swcolson@iu.edu, mmandiwa@iu.edu
joecool@iu.edu, lulacayo@iu.edu
quecox@iu.edu, stfashir@iu.edu
jacuau@iu.edu, pvinod@iu.edu
aadidogr@iu.edu, anthoang@iu.edu
siqidong@iu.edu, clschel@iu.edu
wdoub@iu.edu, ghyatt@iu.edu
dud@iu.edu, sabola@iu.edu
marbelli@iu.edu, esmmcder@iu.edu
moesan@iu.edu, peschulz@iu.edu
jfearri@iu.edu, fimitch@iu.edu
tyfeldm@iu.edu, marystre@iu.edu
tflaa@iu.edu, jolawre@iu.edu
lgflynn@iu.edu, sl92@iu.edu
megofolz@iu.edu, saseiber@iu.edu
bgabbert@iu.edu, jl263@iu.edu
ogift@iu.edu, webejack@iu.edu
jegillar@iu.edu, sjvaleo@iu.edu
mgorals@iu.edu, vilokale@iu.edu
davgourl@iu.edu, cartmull@iu.edu
kynogree@iu.edu, avpeda@iu.edu
qugriff@iu.edu, akundur@iu.edu
aguarda@iu.edu, nireilly@iu.edu
jonhick@iu.edu, danwils@iu.edu
qhodgman@iu.edu, jyamarti@iu.edu
hollidaa@iu.edu, jonvance@iu.edu
tifhuang@iu.edu, arykota@iu.edu
rythudso@iu.edu, parkecar@iu.edu
huntbri@iu.edu, leplata@iu.edu
bradhutc@iu.edu, dy11@iu.edu
jackssar@iu.edu, rair@iu.edu
ajarillo@iu.edu, snuthala@iu.edu
ljianghe@iu.edu, wuyul@iu.edu
coldjone@iu.edu, zhaozhe@iu.edu
dilkang@iu.edu, jorzhang@iu.edu
pkasarla@iu.edu, lenonti@iu.edu
katzjor@iu.edu, rpmahesh@iu.edu

dk80@iu.edu, dwinger@iu.edu
eddykim@iu.edu, roelreye@iu.edu
nklos@iu.edu, jcpilche@iu.edu
delkumar@iu.edu, mvanworm@iu.edu
ekkumar@iu.edu, wallachl@iu.edu
aladkhan@iu.edu, criekki@iu.edu
jl335@iu.edu, ajneel@iu.edu
aalesh@iu.edu, alexschu@iu.edu
jacklian@iu.edu, yz145@iu.edu
georliu@iu.edu, sousingh@iu.edu
klongfie@iu.edu, voram@iu.edu
lopeal@iu.edu, dloutfi@iu.edu
jasoluca@iu.edu, dpepping@iu.edu
imalhan@iu.edu, gsprinkl@iu.edu
kalomart@iu.edu, aw149@iu.edu
crfmarti@iu.edu, shethsu@iu.edu
rmatejcz@iu.edu, bensokol@iu.edu
imaychru@iu.edu, ornash@iu.edu
nkmeade@iu.edu, svaidhy@iu.edu
hmerrit@iu.edu, greymonr@iu.edu
omilden@iu.edu, zisun@iu.edu
jmissey@iu.edu, jvalleci@iu.edu
almoelle@iu.edu, elyperry@iu.edu
nmonberg@iu.edu, mostrodt@iu.edu
isarmoss@iu.edu, jwescott@iu.edu
masmuell@iu.edu, cjromine@iu.edu
cmulgrew@iu.edu, johwarre@iu.edu
arnpate@iu.edu, bpoddut@iu.edu
patel88@iu.edu, bashih@iu.edu
yasmpate@iu.edu, sjxavier@iu.edu
btpfeil@iu.edu, caldsmit@iu.edu
caitreye@iu.edu, alescarb@iu.edu
schwani@iu.edu, edshipp@iu.edu
shahneh@iu.edu, ttieu@iu.edu
fshamrin@iu.edu, mw154@iu.edu
tshore@iu.edu, macsvobo@iu.edu
sowmo@iu.edu, swa5@iu.edu
vthakka@iu.edu, yijwei@iu.edu
davthorn@iu.edu, kvanever@iu.edu