CS146 Fall 2024 – Midterm 1 Solution

Name: _____ **Section: A**

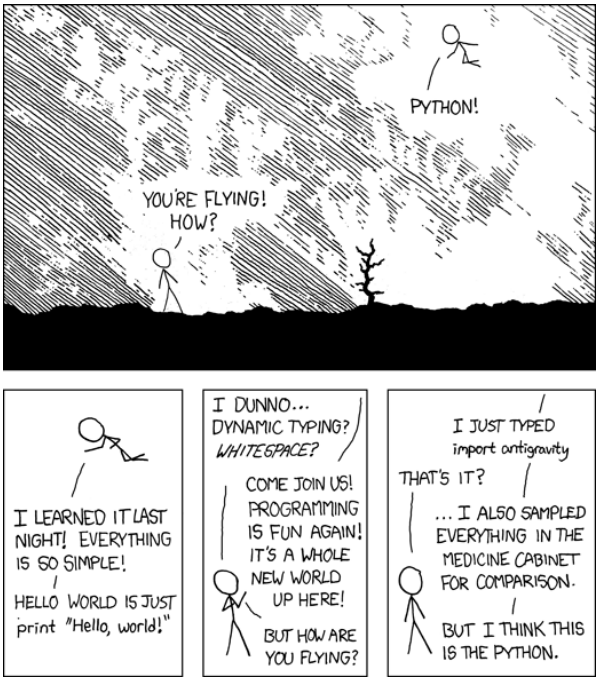
Date: _____ **Start time:** _____ **End time:** _____

Honor Code:

Signature: _____

This exam is closed book, closed notes, closed computer, closed calculator, etc. You may only use (1) the midterm 1 "cheat sheet" from the course page, and (2) a single double-sided letter sheet of notes of your own creation. **You have 2.5 hours.** Read the problem descriptions carefully and write your answers clearly and *legibly* in the space provided. Circle or otherwise indicate your answer if it might not be easily identified. You may use extra sheets of paper, stapled to your exam, if you need more room, as long as the problem number is clearly labeled and your name is on the paper. If you attached extra sheets indicate on your main exam paper to look for the extra sheets for that problem.

You do need to include module imports (if relevant for your code), but do not need to include comments, docstrings or constants in your code.



Question 1: Understanding the role of function scope

Consider the following Python code:

```
w = 1
x = 2
y = 3
z = 4

def bar(x, y):
    z = x + y
    return z*y

def baz(x, z):
    return bar(x, z)

def foo(y, z):
    x = y + 2*z
    return x - 1

w = foo(baz(x, y), baz(x, y))
```

(a) After the above is executed with the green arrow in Thonny, what are the values of `w`, `x`, `y`, and `z`?

i. `w`: **44**

ii. `x`: **2**

iii. `y`: **3**

iv. `z`: **4**

(b) If the last line (and only the last line) was deleted and replaced with

```
def qux(w, x, y, z):
    w = foo(baz(x, y), baz(x, y))
    return w
x = qux(w, x, y, z)
```

what are the values of `w`, `x`, `y`, and `z` if the code was executed with the green arrow in Thonny?

i. `w`: **1**

ii. `x`: **44**

iii. `y`: **3**

iv. `z`: **4**

Question 2: Writing functions with randomness

Write a function named `roll_dice`, that takes one parameter, `faces`, and prints the sum of rolling 2 independent dice each with `faces` number of faces. The first dice is labeled with faces 1 to `faces`, inclusive and the second with the next `faces` numbers. For example, if `faces` was 6, the first dice is labeled 1-6 to the second 7-12. Your function should not return a value. Here are some sample calls and output.

```
>>> roll_dice(6)
11
>>> roll_dice(1)
3
```

Solution:

```
import random

def roll_dice(faces):
    return random.randint(1,faces) + random.randint(faces+1, 2*faces)
```

Question 3: Writing functions with loops

Write two functions, one using a `for` loop and the other using a `while` loop, to print the numbers between 1 to 100 inclusive, in ascending order, one number per line, that are multiples of both 3 *and* 4. Your functions should not have any parameters and should not return a value. Your solution will be exclusively evaluated on correctness.

(a) Using a `for` loop

Solution:

```
def forloop():
    for i in range(12, 101, 12):
        print(i)
```

Or perhaps more generically (but less efficient);

```
def forloop():
    for i in range(1, 101):
        if i % 3 == 0 and i % 4 == 0:
            print(i)
```

(b) Using a `while` loop

Solution:

```
def whileloop():
    i=12
    while i <= 100:
        print(i)
        i += 12
```

Or perhaps more generically (but less efficient);

```
def whileloop():
    i=1
    while i <= 100:
        if i % 3 == 0 and i % 4 == 0:
            print(i)
        i += 1
```

Question 4: Choosing appropriate loops

For each of the following tasks, indicate whether a `for` loop or a `while` loop would be more appropriate. Briefly explain your choice.

(a) One approach to generating secure random numbers is to ask the user to move the mouse, recording the least significant digits of the mouse position, until sufficient randomness has been obtained (the amount of movement required depends on the position values).

◯ `for` loop ✓ **`while` loop**

> **Solution:** Since we don't know how much movement is required to achieve sufficient randomness, a `while` loop is most appropriate.

(b) Have the user move their mouse to four different positions on a drawing application's canvas to calibrate the mapping between the mouse position and the canvas position.

✓ **`for` loop** ◯ `while` loop

> **Solution:** Since we need click on a fixed number of positions, known before the loop starts, a `for` loop is more appropriate.

(c) DNA can represented as a string containing the letters A, C, G, and T. A function to count the fraction of G's and C's in a DNA string provided as an argument.

✓ **`for` loop** ◯ `while` loop

> **Solution:** Since we need to examine all letters in the string and know the length of the DNA string prior to the loop starting, a `for` loop would be most appropriate.

Question 5: Creating simpler equivalent conditionals

The function below has two integer parameters, `a` and `b`. The function works as desired, however, it is very verbose. Rewrite the function to have identical behavior (i.e., for all possible values of `a` and `b` return the same value), but to be as concise as possible.

```
def could_be_better(a, b):
    if a > 0:
        if b >= 0:
            return True
        elif b < 0:
            return False
    elif a <= 0 or b < 0:
        return False
    else:
        return True
```

Solution:

```
def better(a, b):
    return a > 0 and b >= 0
```

Question 6: Finding errors

The following function was designed to return a binary message (containing only "0"s and "1"s) with a parity bit appended. If the message has an odd number of "1"s, a "1" is appended to make the total count even. If there are an even number of "1"s (or zero), a "0" is appended (the keep the count even). The examples to the right show the intended behavior. There are 3 problems with this code. Report the line number of and briefly describe: i) one syntax error, ii) one runtime error (syntactically valid Python that generates an error when actually executed) and iii) one logical error (the code would execute to completion if the other errors are fixed but produces incorrect results), for three errors total. The errors should not be variations of the same issue and should impact correctness, not just style. The implementation can assume all inputs are strings only containing "0" or "1". You do not need to fix the errors.

```
1 def parity(msg):
2     ones = 0
3     for bit in range(msg):
4         if bit != "0":
5             ones += 1
6     if ones % 2 = 1:
7         return msg + "0"
8     else:
9         return msg + "1"
```

```
>>> parity("000")
'0000'
>>> parity("001")
'0011'
>>> parity("011")
'0110'
```

(a) Syntax error on line: **6**

> **Solution:** We can't use the assignment operator (`=`) as a relation, we should use `==`.

(b) Runtime error on line: **3**

> **Solution:** `range` expects integer arguments, not a string. A correct loop would be:
> `for bit in msg:`

(c) Logical error on line: **6**:

> **Solution:** The relation is True when the count of 1s is odd and so the incorrect parity digit is appended. Line 6 should be `if ones % 2 == 0` or lines 7 and 9 swapped. An alternate logical error that was accepted assumes that `bit` remains an integer (instead of being each digit in the string). In that case, the code runs, but the conditional expression on line 4 is always `True`, incorrectly treating every digit as a "1".

Question 7: Writing functions with sequences

Python lists have a very helpful `remove` method that removes the first occurrence of a value in a list. Unfortunately there isn't a similar method for strings. Write a function named `remove`, with two parameters, a sequence `seq` and `item`, which returns a copy of `seq` with the first occurrence of `item` removed. If there is more than one occurrence, only the first should be removed. Your function should work for lists and strings as the `seq` argument, and any type for the `item` parameter. *You are not permitted to use `isinstance` or other type-related functions.* As a hint, slicing evaluates to a copy of a sequence of the same type and equality operators can be applied to values of different types. Some examples are shown below on the left, and, for reference, some relevant string/list comparisons are shown on the right. Your solution will be exclusively evaluated on correctness.

```
>>> remove("abcdec", "c")
'abdec'
>>> remove("abcdec", "f")
'abcdec'
>>> remove([1, 2, 3, 4, 3], 3)
[1, 2, 4, 3]
>>> remove("abcde", 1)
'abcde'
```

```
>>> "a" == "a"
True
>>> "a" == 1
False
>>> 1 == 1
True
```

Solution:

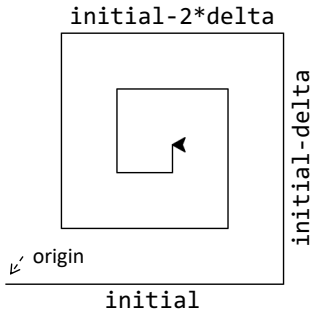
```
def remove(seq, item):
    for i in range(len(seq)):
        if seq[i] == item:
            return seq[:i] + seq[i+1:]
    return seq
```

Question 8: Utilizing turtle and other modules

Use the lines below to construct a function named `spiral` that takes two parameters, `initial`, and `delta`, and draws a right-angle spiral whose initial edge has length `initial`, and each subsequent edge decreases by `delta` (like shown in the figure below). Assume that the turtle (the pen) is initially at the origin (as indicated in the figure) and facing to the right. The spiral should end if the edge length is not greater than 0, with the turtle at the last drawing position.

Write the line number and the *correctly indented* line of code in the rows below. The first line is done for you as an example. You may only use each of the scrambled lines once. Not all scrambled lines or rows may be needed. If there is an inconsistency between the line number and the code, the latter will be used as your answer.

```
1       side = initial
2       for i in range(delta):
3   import turtle as t
4           side = i * delta
5           side = side - delta
6       while side > 0:
7   def spiral(initial, delta):
8           t.forward(side)
9           t.left(90)
10          t.right(90)
11      for side in range(0, initial, delta):
```




```
>>> spiral(200, 20)
```

Line number	Code
3	`import turtle as t`
7	`def spiral(initial, delta):`
1	`    side = initial`
6	`    while side > 0:`
8	`        t.forward(side)`
9	`        t.left(90)`
5	`        side = side - delta`

Page intentionally left blank.