C200 Replacement for Assignment 3 or Assignment 4
Functions, Containers, Choice, Bounded Loops
FALL 2022

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

Indiana University, Bloomington, IN, USA

October 9, 2022

We're now at a juncture–the midterm is next week with neither laboratory or homework. I've created this **optional** homework to replace the lowest score from either Assignment 3 or Assignment 4. For this optional homework, please select three problems from the six. Since this is optional, we cannot assign partners; however, you are allowed to have a partner (of your own choosing) for this, but you must make explicit on the homework who your partner is. You'll see this on the *.py file as a comment–please fill this in. To give you ample time this won't be due until Sunday, October 16, 2022 10:59PM EST. There isn't a unit test or autograder for this, since I've made it outside of normal homework, but you should be comfortable now with determining if your implementations are adequate. Have a great Fall Break!

Problem 1: Factorial and a single, bounded loop

The factorial function is used in many problems. Factorial is a post-fix function that returns the product of numbers from n to 0. Here is the function for some values:

$$0! = 1 \tag{1}$$
$$1! = 1 \tag{2}$$
$$2! = 2 \times 1 \tag{3}$$
$$3! = 3 \times 2 \times 1 \tag{4}$$
$$4! = 4 \times 3 \times 2 \times 1 \tag{5}$$

One way of thinking about factorial is choice! Suppose you have n different things to choose from like this list ['a','b','c']. You have to keep choosing until there's nothing left. Here are the different choices:

```
abc
acb
bac
bca
cab
cba
```

There are six or 3!. The reason 0! = 1 is without any things to choose from, there's only one choice. The math module has the factorial function:

```
>>> import math
>>> math.factorial(0)
1
>>> math.factorial(4)
24
```

Implement factorial. Here is the code you'll use:

```
def factorial(n):
    pass

for i in range(6):
    print(f"{i}! = {factorial(i)}")
```

with output:

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
```

Problem 2: Lists and a single, bounded loop

Given two lists of numbers of the same length, return a list that has the greater of the numbers at the same location. For example, lst1=[x0,x1,...,xn] and lst2 = [y0,y1,...,yn], then the function gl(lst1,lst2) returns [z0,z1,...,zn] where z0 is x0 if x0 $\geq$ y0 otherwise y0. Here is the code:

```
ef gl(lst1,lst2):
    pass

```

```
4 data = [[[1,0,0,1],[0,1,1,0]],[[],[]],[[1,2,3,4],[5,4,3,2]]]
5
6 for x,y in data:
7     print(f"{x,y} {gl(x,y)}")
```

has output

```
1 ([1, 0, 0, 1], [0, 1, 1, 0]) [1, 1, 1, 1]
2 ([], []) []
3 ([1, 2, 3, 4], [5, 4, 3, 2]) [5, 4, 3, 4]
```

You can solve this using bounded loops, but if you'd like you may also use a while loop. Remember we had said that sometimes subscripting is better suited for a problem than using an iterator. If you want, you may search InScribe for a post titled **Using in** to revist the difference between an iterator and a subscriptor. Also, I'd like to remind you of using more than one index variable if the iterable has the same number of objects. Let's look at a couple of examples first:

```
1 >>> for x,y in [[1,2],[3,2],[5,1]]:
2 ...     x,y
3 ...
4 (1, 2)
5 (3, 2)
6 (5, 1)
```

There are two index variables, x and y. Each member of the iterator [[1,2],[3,2],[5,1]] has two objects. The first member is assigned to x and the second member is assigned to y. In fact, Python allows you to have a container as an index variable when the size matches the object sizes:

```
1 >>> for (x,y) in [[1,2],[3,2],[5,1]]:
2 ...     x,y
3 ...
4 (1, 2)
5 (3, 2)
6 (5, 1)
7 >>> for [x,y] in [[1,2],[3,2],[5,1]]:
8 ...     x,y
9 ...
10 (1, 2)
11 (3, 2)
12 (5, 1)
```

Since the problem has two objects the same length, you can, if you want, take advantage of that in your program.

Problem 3: e^x using a loop

The math module has a function to find e^x for some value of x as math.exp(x). This value is actually from a summation:

$$e^x = x^0/0! + x^1/1! + x^2/2! + \cdots \tag{6}$$

We obvious can't do an infinite loop so we decide to stop at some value n:

$$e^x = x^0/0! + x^1/1! + x^2/2! + \cdots + x^n/n! \tag{7}$$

which has $n+1$ terms (hint: range). Since you've implemented factorial, you can now implement a function that computes e^x. Here's the code:

```
1 import math
2
3 def my_e(n,num_terms):
4     pass
5
6 print(math.exp(5))
7 print(my_e(5,16))
```

here's the output:

```
1 148.4131591025766
2 148.41021027504306
```

As you can see, with 17 values our function is becoming exactly like the math module.

Problem 4: Substring using two loops (nested)

Given a non-empty string s , implement the function ss(s) that returns a dictionary of all the proper substrings with the count the number of times the substring appears. One approach is to determine the strings of length 1,2,3,. . ., n-1. For each length, start at the first position, the second position, and so on. Since this is a **proper** substring, the original string isn't included. For example,

```
1 def ss(x):
2     pass
3
4 data = ["s","abc","abcabc","aa"]
5 for d in data:
6     print(d)
7     print(ss(d))
```

has output:

```
s
{}
abc
{'a': 1, 'b': 1, 'c': 1, 'ab': 1, 'bc': 1}
abcabc
{'a': 2, 'b': 2, 'c': 2, 'ab': 2, 'bc': 2, 'ca': 1, 'abc': 2, 'bca': 1, 'cab': 1, 'abca': 1, 'bcab': 1, 'cabc': 1, 'abcab': 1, 'bcabc': 1}
aa
{'a': 2}
```

Problem 5: Replacement using two loops

Suppose you have a list of lists and two objects old, new. Return the list of lists after replacing the old with new object's value. Here's the code you'll use:

```
def lst_replace(old,new,lst):
    pass

print(lst_replace(1,"dog", [[1,2],[2,[2],1],[],[1,1,]]))
print(lst_replace(2,"dog", [[1,2],[2,[2],1],[],[1,1,]]))
print(lst_replace([2],"dog", [[1,2],[2,[2],1],[],[1,1,]]))
print(lst_replace(1,"dog", []))
```

has output:

```
[['dog', 2], [2, [2], 'dog'], [], ['dog', 'dog']]
[[1, 'dog'], ['dog', [2], 1], [], [1, 1]]
[[1, 2], [2, 'dog', 1], [], [1, 1]]
[]
```

Problem 6: Remove these silly brackets using while

Given a list of any number of lists, foo is a function that puts everything in a single list in the same order. We'll need to determine if an object is a list–we can use isinstance(x,list) (see lecture notes) that returns true if x is a list and false otherwise. Here the function:

```
def foo(lst):
    pass

```

```
data = [[[1],2,[[3]]], [[1,[2,[3]],4,[[5]],6,[7,8]]],
        [1,2,3,[[[4]]]],[],[1,2,3]]

for d in data:
    print(f"{d} \n {foo(d)}")
```

with output:

```
[[1], 2, [[3]]]
 [1, 2, 3]
[[1, [2, [3]], 4, [[5]], 6, [7, 8]]]
 [1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, [[[4]]]]
 [1, 2, 3, 4]
[]
 []
[1, 2, 3]
 [1, 2, 3]
```

A hint: you'll have to have a temporary list. If you have a list [x,y,...] and x is a not a list, then you can append x directly to the temporary list. If x is a list, you'll have to work on x + [y,...].