CS 301: Practice Problems

These exercises will help you improve your understanding of operators (arithmetic and logical), boolean logic, binary numbers and pseudocode.

If you have trouble understanding or solving the problems, be sure to stop by during office hours!

Pseudocode

You have already been introduced to the concepts of Pseudocode, Control flow and State. Solve the following problems (instructions similar to the ones in the worksheet given in class).

Problem 1

State:

Inp:	8	Val:	1	Out:	

Code:

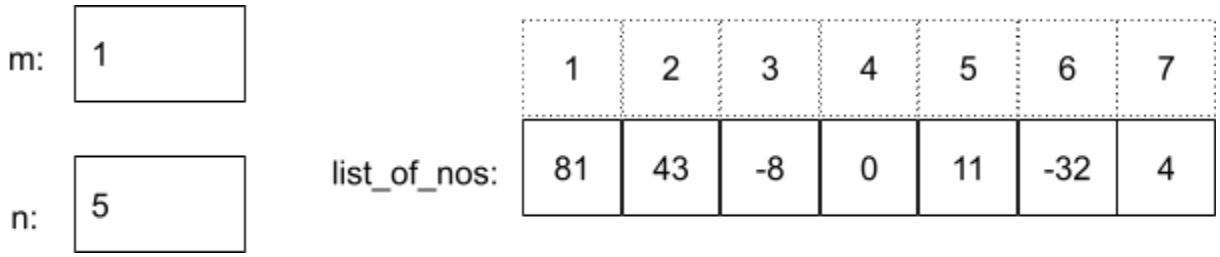
1. If ‘Inp’ is greater than 0, go to step 2. Otherwise, go to step 5.
2. Add the value in ‘Val’ to the value in ‘Inp’. Store this in ‘Val’.
3. Subtract 1 from the value in ‘Inp’.
4. Go to step 1.
5. Copy the value in ‘Val’ to the value in ‘Out’.

Questions:

1. What is the final value stored in ‘Out’?
2. What is the objective of the program? If necessary, try doing the above again with a different value of Inp (eg. 5, 6, etc.) (Hint: The program is designed while assuming that Inp is a positive number!)
3. Instead of the above steps, can you come up with a one step ‘Program’ which can do the same task? (Hint: Do you know a formula for what the program is trying to achieve?)

Problem 2

State:



list_of_nos is a list of numbers. The numbers in the dotted list above list_of_nos corresponds to the 'index'(position) of each individual number in the list_of_nos. Eg. Index of -32 is 6, index of 43 is 2, etc.

The element with index 'm' is referred to as the m^{th} element below.
Code:

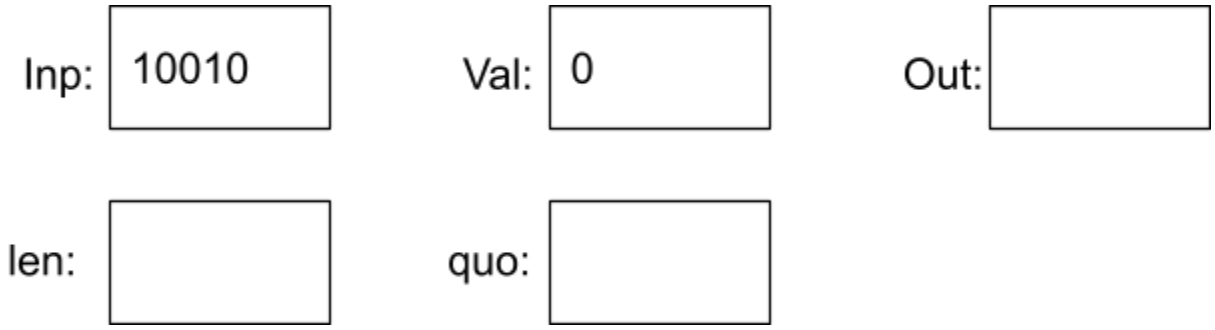
1. Add the n^{th} element of list_of_nos to the m^{th} element of list_of nos and store this value in the m^{th} element of list_of_nos.
2. Subtract the value of the n^{th} element of the list from the m^{th} element and store it in the n^{th} element of list_of_nos.
3. Subtract the value of the n^{th} element of the list from the m^{th} element and store it in the m^{th} element of list_of_nos.

Questions:

1. What are the final contents of list_of_nos?
2. Repeat the above process when m and n are 3 and 5 respectively.
3. What is the objective of the program? If necessary, try doing the above again with more values of m and n (Hint: m and n need to be between 1 and 7 for the program to work. Why?)
4. What happens when m is equal to n? Does the program work as intended?
5. Can you think of an easier way to achieve the same objective? You can define a new state if needed.

Problem 3

State:



Code:

1. Store the **length** of the value in the ‘Inp’ box in the ‘len’ box.
2. Subtract 1 from the value in ‘len’ and store it in ‘len’.
3. Divide the value in ‘Inp’ by 10**’len’. Store the quotient in ‘quo’. (** is the exponent operator in python, eg. 10**3=1000)
4. Find the remainder when ‘Inp’ is divided by 10**’len’. Store this value in ‘Inp’.
5. Multiply the value in ‘Val’ by 2 and add ‘quo’ to it. Store this in ‘Val’.
6. If the value in ‘len’ is 0, go to step 7. Otherwise, go to step 2.
7. Set the value of ‘Out’ to be equal to the value in ‘Val’.

Questions:

1. What is the final value stored in ‘Out’?
2. What is the objective of the program? If necessary, try doing the above again with a different value of Inp (eg. 110, 1011, etc.) (Hint: The program is designed while assuming that Inp is in binary!)

Arithmetic and Comparison Operators

Operators are symbols which help in the manipulation of values.

For a detailed overview of operators, try reading:
https://www.tutorialspoint.com/python/python_basic_operators.htm

Arithmetic operators:
+(Add), -(subtract), *(multiply), /(divide), //(integer divide), %(modulus/remainder), **(exponent)
Arithmetic operators are very simple, just math! Try running the following arithmetic expressions in python to see if the output matches what you would expect it to be. Try solving them first, and then run on python to compare your answers.

```
a. 5 * 7 (to run in python run the following code snippet: print(5*7))
b. 1 + 4
c. 77 - 65
d. 3 ** 4
e. 18 // 4 - 4 * 3
f. 7 / 5
g. 7 // 5
h. 7 + 5 * 9
i. (7 + 5) * 9
j. 2 * 7 - 8 + 3 ** 2
k. 2 * 7 - (8 + 3) ** 2
l. 2 * (7 - 8) + 3 ** 2
```

To check your answers using python, you can do either of these two:

I. Type the below expressions in directly into a python console (OR)
II. Place each expression inside the parentheses of a print() statement in a python script and run the script!

Did the outputs match your answers? Make sure you follow the correct order of operations (**P**arentheses, **E**xponent, **M**ultiplication, **D**ivision, **A**ddition, **S**ubtraction).
This is the concept of **Operator Precedence** in python. Python follows the same rules that we learned in elementary math!

Comparison Operators: == (is it equal to?), > (greater than), < (less than), etc.
These operators help compare two or more variables/numbers. Try out the following expressions on python (Hint: Each of the below expressions has a True/False answer)

a. `7 == 8`
b. `10 == 10`
c. `6 == 8 - 2`
d. `4 > 7`
e. `5 + 3 <= 10 - 2` (What is this operator? Check out the link listed on the previous page!)
f. `"Jack" == "Hack"` (Yes, we can use them to compare strings too!)
g. `"Jack" == "Jac"+"k"`

Note: Arithmetic operators have greater precedence than comparison operators.
Comparison operators return the data type of Boolean (which can hold either True or False).

Boolean logic and Logical Operators

Using boolean logic, we can evaluate multiple conditions at once. The three fundamental operators used are **and**, **or** and **not**. These are called **Logical Operators**.

Operator **and** is used to check when both the conditions are true. For example,
`(5>1) and (3<5)` will return `true` because both (5>1) and (3>5) are individually true.
`(5>1) and (7<5)` will return `false` as (7<5) is false.

Operator **or** is used to check when at least one of them is true. For example,
`(5>1) and (3<5)` will return `true` because both (5>1) and (3>5) are individually true.
`(5>1) and (7<5)` will return `true` even though (7<5) is false.
`(5>10) and (7<5)` will return `false`. Can you explain why?

Operator **not** simply negates the boolean value of the expression. For example,
`not (3>6)` will return `true` as (3>6) is false. `not (13>6)` will return `false` as (13>6) is true.

Now try solving these boolean logic expressions on your own:

a. `(7 == 8) or (9 < 100 / 20)`
b. `not (11 == 13 - 28 / 14)`
c. `(-8 > -7) and (12 > 2)`
d. `not (-10 > -11) or (12 >= 10)`
e. `not ((15 > 4) or (11 < 2**4))`
f. `not (5 > 10) and (11 == 23 % 12) or ("Jack" == "Jill")`

Note: The order of precedence of logical operators is: not, and, or (not has highest precedence). Keep that in mind while solving the above.
Verify your answers by running them on python!

Binary Numbers (Optional)

*Note: You **won't** be tested on binary numbers, this is purely for your knowledge.*

Computers only understand ones and zeros. Thus we need the concept of binary numbers!

The number system we are most familiar with is the Decimal system. It is a base-10 system. Do you know why decimal number system is called a base-10 system?

The binary number system is a base-2 system. The only two symbols are 1's and 0's.

Do you know how to convert from binary to decimal and vice-versa?
Read these if you are not clear how to go about it:
https://www.rapidtables.com/convert/number/binary-to-decimal.html
https://www.rapidtables.com/convert/number/decimal-to-binary.html

Let's practice converting, try filling in the blanks:

Binary	Decimal
101	
	101
	32
10101	
	3
	63
1100	

Check your answers using Python!

Note: Binary numbers in python are preceded by '0b'.
Convert from decimal to binary, run: print(bin(<your number>)). Eg: print((bin(2))) will return 0b10.
Convert binary to decimal: eg. print(0b10) will return 2. print(0b101) will return 5.