Faculty of Mathematics
Waterloo, Ontario N2L 3G1

Centre for Education in
Mathematics and Computing

Grade 7/8 Math Circles

October 28, 2020

Propositions and Control Flow - Solutions

Introduction

Computer Science is a continually growing field. Every day, people around the world use technology to access information, create and relax. In fact, you are probably using a computer of some form to read this document! Mathematics provides the backbones for Computer Science. This week, we will look at one such example. We will first talk about Propositions and their role in Computer Science. Then, we will use that to start to create our own programs in Python! First, what is a Proposition?

Definition 1. *Proposition*
A proposition, is a statement that can be evaluated to one of true or false. For example, "it is raining outside" is a proposition as it can be checked and will either be true or false.

Definition 2. *Truth Value*
A truth value is whether a proposition is true or false. For example, the truth value for the proposition "$5 + 2 \geq 8$" is false.

Logical Operators

Definition 3. *Logical Operator*
A logical operator joins one or more propositions to create a larger proposition.
Examples: AND, OR, NOT

Definition 4. *AND Operator*
Takes two propositions and gives a "true" truth value if both propositions are true. Otherwise, if gives a "false" truth value. The AND operator is represented by $\wedge$. Ex. $P \wedge Q$

Definition 5. *OR Operator*
Takes two propositions and gives a truth value of "true" if either (or both) of the propositions are true. Otherwise, if gives a "false" truth value. The OR operator is represented by $\vee$. Ex. $P \vee Q$

Definition 6. *NOT Operator*
Takes one proposition and gives the opposite valuation. If the proposition is "true" it yields "false" and vice-versa. The NOT operator is represented by $\neg$. Ex. $\neg P$

Note that our *OR* operator is different from the English or. For example, if the proposition is "I ate the cake or I ate ice cream" it would be true if I ate only the ice cream, only the cake or both the cake and the ice cream.

Simple and Compound Propositions

Notice that, in our examples for the definitions above, we substituted propositions for letters. This is a common way to simiplify our propositions and find general rules for propositions.

Example. "It is not raining and I went outside" and "I do not like broccoli and I ate carrots" could both be represented by $(\neg P) \wedge Q$.

P represents "It is raining" or "I like broccoli"
Q denotes "I went outside" or "I ate carrots"

If we look only at the general proposition, we know that if P is false and Q is true, the proposition is true. Then, without looking at the two English propositions, we can know that they are true if their corresponding P is false and corresponding Q is true.

So, we can convert between English propostions and propositions using letters as substitutes. To help with consistency, we make the following definitions:

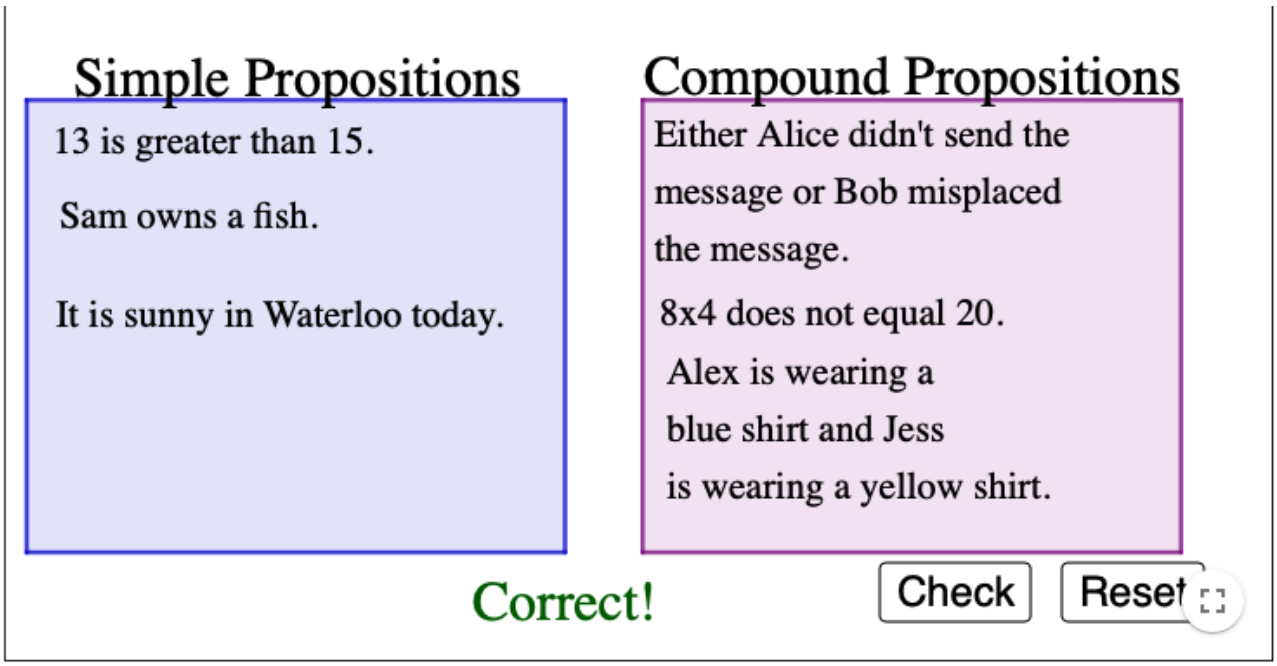
Definition 7. *Simple Proposition*
A proposition that has not been formed with the use of logical operators.
Example: "I walked my dog today."

Definition 8. *Compound Proposition*
A proposition that has been formed with the use of logical operators.
Example: "It is sunny today and I walked my dog today".

Compound propositions can be broken into a combination of simple propositions and logical operators. For the example above, "It is sunny today" and "I walked my dog today" are both simple propositions and "and" is a logical operator.

Exercise 1. Practice distinguishing between simple and compound propositions here: *https://www.geogebra.org/m/yzz3wym5*.

Solution:



When we convert propositions in English to one using letters, we substitute simple propositions with letters and leave logical operators. It is helpful to add brackets to establish an order of precedence. What if I had:

"I did not walk my cat and I walked my dog"

Then, this would translate into $\neg P \wedge Q$, where P is the proposition "I did walk my cat" and Q is the proposition "I walked my dog". Does this mean $\neg(P \wedge Q)$ or $(\neg P) \wedge Q$? We know from the English that we want $(\neg P) \wedge Q$, as we are saying that I walked my dog but not my cat, not that I didn't walk both my cat and my dog. We use brackets to establish this and thus represent the situation as $(\neg P) \wedge Q$.

Truth Valuations

Since we now have general propositions, it can be helpful to know when they would evaluate to true and when they would evaluate to false.

Definition 9. *Truth Valuation*
A truth valuation is an assignment of truth values to all simple propositions in a proposition. For example, a truth valuation of $P \wedge Q$ would be P is true and Q is false.

Typically, we want to see all possible truth valuations of a given proposition. Truth tables offer a concise way of showing these valuations and the resultant truth values. Often, in a truth table, we use 1 to represent "true" and 0 to represent "false". First, we need to list all possible truth valuations. The number of possible truth valuations depends on the number of simple propositions:

One Simple Proposition:

P
1
0

Two:

P	Q
1	1
1	0
0	1
0	0

Three:

P	Q	R
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	1
0	0	0

Three is the highest number of simple propositions that we will need this week. Once we have listed all the possible truth valuations, we list the resulting truth value of the final proposition, given the truth valuation in that row.

As an example, below are the truth tables of the logical operators:

Truth Table for $\wedge$:

P	Q	$P \wedge Q$
1	1	1
1	0	0
0	1	0
0	0	0

Truth Table for $\vee$:

P	Q	$P \vee Q$
1	1	1
1	0	1
0	1	1
0	0	0

Truth Table for $\neg$:

P	$\neg P$
1	0
0	1

Truth tables aren't always easy to fill out. What if I had $(P \wedge Q) \vee ((\neg Q) \vee P)$? Watch this video: *https://youtu.be/XSl4VgUghgI* to learn how to fill out a larger truth table.

Exercise 2. Practice filling out this truth table: *https://www.geogebra.org/m/yx4vhutc*.

Solution:

P	Q	$P \vee Q$	$\neg P$	$Q \wedge (\neg P)$	$(P \vee Q) \wedge (Q \wedge (\neg P))$
1	1	1	0	0	0
1	0	1	0	0	0
0	1	1	1	1	1
0	0	0	1	0	0

Correct!

Check Reset

Introduction to Python

In order to create the apps and programs that we use to interact with our devices, Computer Scientists use programming languages. It is through these languages that we tell computers how to behave. One of the key aspects when coding is the use of boolean expressions. This week, we will be taking what we have learned about propositions and use them to create Python programs. First, we need to learn some basic coding skills.

Watch this video: *https://youtu.be/wSEsKx-p_kU* to learn the basics of coding using the Python programming language. Throughout the video, there will be places where you are asked to pause and try some exercises to practice. Those exercises are listed below. All exercises asking you to write a program can be done using this panel: *https://open.cs.uwaterloo.ca/python-from-scratch/python-panel/*, which allows you to run the code that you write. Write your code in the top box and click *Run* to see what your program outputs in the bottom box.

Exercise 3. Write a program that outputs *Hello World!*

Solution:

```
print("Hello World!")
```

Exercise 4. Write a program that, using only one variable and only using print statements with variables, prints the following, one word at a time: *Hi Human! How are you?*

Solution:

```
1  word = "Hi"
2  print(word)
3
4  word = "Human!"
5  print(word)
6
7  word = "How"
8  print(word)
9
10 word = "are"
11 print(word)
12
13 word = "you?"
14 print(word)
15
16
```

Exercise 5. Write a program with four variables, *one, two, three* and *four*, which prints out the information in each variable in the following order: *three, one, four, two.*

Solution:

```
one = 1
two = "Hi!"
three = True
four = 3

print(three)
print(one)
print(four)
print(two)
```

Exercise 6. Write a program that has a variable x which is equal to some integer. It then performs the operations listed below on x. In each step, take the result from the step before as the input.

1. Multiply by 8.
2. Add 16.
3. Divide by 4.
4. Find the remainder when divided by 2.
5. Add 16.
6. Subtract 3.

Example: $x = 10$

1. $8 * 10 = 80$
2. $80 + 16 = 96$
3. $96/4 = 24$
4. $24\%2 = 0$
5. $0 + 16 = 16$
6. $16 - 3 = 13$

Your program should output 13.0, no matter what integer x is initially equal to.

Solution:

```
x = 10

x = x * 8
x = x + 16
x = x / 4
x = x % 2
x = x + 16
x = x - 3

print(x)
```

Exercise 7. Write a program, similar to that in Exercise 7, that always outputs your favourite number.

Solution: This program always outputs 132.0.

```
1   x = 15
2
3   x = x * 4
4   x = x + 13
5   x = x % 2
6   x = x + 99
7   x = x / 5
8   x = x * 7
9   x = x - 8
10
11  print(x)
```

Control Flow

Now that we know the basics of Python, we can begin to combine what we learned about propositions. In Computer Science, we often need to chose whether to perform an action based on given information. To do this, we use boolean expressions, which are very similar to propositions. Like propositions, boolean expressions are either true or false. Our logical operators remain the same, but rather than using phrases like "it is raining", we use expressions like $x > 5$.

Definition 10. *Control Flow*
Control flow is the order code is executed. When writing programs, we don't always want to go exactly in the order of the lines listed. To change the order, we use tools like if..else blocks and while loops. To use these, we need boolean expressions.

Definition 11. *Comparision Operators*
To check if a variable is related to a number or another variable, we often use comparision operators. The comparison operators are: ==, !=, >, <, >= and <=. The result of *a comparison operator b* is always either true or false. $a == b$ yields true when a and b are equal and a != b is true otherwise. $a > b$ and $a < b$ gives a true truth value if a is strictly greater (or less for <) than b. $a >= b$ and $a <= b$ are true when a is greater/less than or equal to b.

Definition 12. *Logical Operators in Python*
To write the logical operators in Python, we use *and, or* and *not()*, where the brackets in *not* contain the portion that we are negating.

Definition 13. *If...else Block*
To use these boolean expressions, we often add them to if...else blocks. An if...else statement contains a boolean expression. If the boolean expression yields a truth value of true, the code in the if section will run. Otherwise, the code in the else section will.

Definition 14. *While Loop*
A while loop will run through the same piece of code over and over until its boolean expression is false.

For an example of if...else blocks and while loops, watch this video:
https://youtu.be/cf531qrw41w

Exercise 8. Write a boolean expression that is only true when x is greater than 5 and y is less than or equal to 7. Write a program to help check your answer.

Solution:
Boolean Expression: $(x > 5)$ and $(y <= 7)$

```
1  x = 6
2  y = 4
3
4  if (x > 5) and (y <= 7):
5      print("True")
6  else:
7      print("False")
```

Exercise 9. Write a program that outputs whether x is an odd or an even number.

Solution:

```
1  x = 3
2
3  if (x % 2) == 0:
4      print("x is even")
5  else:
6      print("x is odd")
```

Exercise 10. Write a program that outputs every second number between two variables, x and y, starting with outputing x. Assume that x is less than y.

Solution:

```
1  x = 2
2  y = 10
3
4  z = x
5  while (z < y):
6      print(z)
7      z = z + 2
8
```

Problem Set

1. Find the truth valuation for each of the given propositions if P is false, Q is true, and R is true.

   (a) $(P \wedge Q) \vee (\neg R)$

   (b) $(Q \wedge (\neg P)) \wedge (R \wedge P)$

   (c) $P \vee (R \wedge Q)$

   (d) $(P \wedge (Q \vee (R \wedge (\neg P)))) \vee (Q \wedge (P \vee R))$

   Solution:

   a) False b) False c) True d) True

2. Find a proposition that is only true when either P and Q are both true or P is not true and R is true.

   Solution:

   $(P \wedge Q) \vee ((\neg P) \wedge R)$

   Note: This is solution is one of many possible.

3. Find a proposition with P and Q that always has a true truth value (no matter the truth valuation).

   Solution:

   $(P \vee Q) \vee ((\neg P) \wedge (\neg Q))$

   Note: This is solution is one of many possible.

4. Find a proposition with P and Q that always has a false truth value.

   Solution:

   $(P \wedge Q) \wedge (P \wedge (\neg Q))$

   Note: This is solution is one of many possible.

5. Draw the truth tables for the following propositions.

   (a) $(P \wedge Q) \wedge (\neg P)$

   (b) $P \wedge (Q \vee (R \vee (\neg Q)))$

   (c) $(P \wedge R) \vee (Q \wedge \neg(Q \wedge P))$

   (d) $((P \vee Q) \vee (R \wedge (\neg P))) \vee (\neg Q)$

   Solution:

   (a)

P	Q	$P \wedge Q$	$\neg P$	$(P \wedge Q) \wedge (\neg P)$
1	1	1	0	0
1	0	0	0	0
0	1	0	1	0
0	0	0	1	0

(b)

P	Q	R	$\neg Q$	$R \vee (\neg Q)$	$Q \vee (R \vee (\neg Q))$	$P \wedge (Q \vee (R \vee (\neg Q)))$
1	1	1	0	1	1	1
1	1	0	0	0	1	1
1	0	1	1	1	1	1
1	0	0	1	1	1	1
0	1	1	0	1	1	0
0	1	0	0	0	1	0
0	0	1	1	1	1	0
0	0	0	1	1	1	0

(c)

P	Q	R	$P \wedge R$	$Q \wedge P$	$\neg(Q \wedge P)$	$Q \wedge (\neg(Q \wedge P))$	$(P \wedge R) \vee (Q \wedge (\neg(Q \wedge P)))$
1	1	1	1	1	0	0	1
1	1	0	0	1	0	0	0
1	0	1	1	0	1	0	1
1	0	0	0	0	1	0	0
0	1	1	0	0	1	1	1
0	1	0	0	0	1	1	1
0	0	1	0	0	1	0	0
0	0	0	0	0	1	0	0

(d)

P	Q	R	$\neg P$	$\neg Q$	$P \vee Q$	$R \wedge (\neg P)$	$(P \vee Q) \vee (R \wedge (\neg P))$	$((P \vee Q) \vee (R \wedge (\neg P))) \vee (\neg Q)$
1	1	1	0	0	1	0	1	1
1	1	0	0	0	1	0	1	1
1	0	1	0	1	1	0	1	1
1	0	0	0	1	1	0	1	1
0	1	1	1	0	1	1	1	1
0	1	0	1	0	1	0	1	1
0	0	1	1	1	0	1	1	1
0	0	0	1	1	0	0	0	1

6. Write a boolean expression that can be used to to determine if both x is less than 5 and y is 6.

   Solution:
   $(x < 5)$ and $(y == 6)$

7. Write a program that will print True if x is between or equal to y and z and x is not a multiple of 5. Assume that y is smaller than z.

   Solution:

```
x = 4
y = 3
z = 5

if (x >= y) and (x <= z) and ((x % 5) != 0):
    print("True")
else:
    print("False")
```

8. Write a program that will print all the numbers between, but not equal to, y and z that are not multiples of 2, but are multiples of 3. Assume you do not know which of y and z are smaller.

   Solution:

```
y = 0
z = 16

if (y >= z):
    curr = z + 1
    while(curr < y):
        if ((curr % 2) == 1) and ((curr % 3) == 0):
            print(curr)
        curr = curr + 1

else:
    curr = y + 1
    while(curr < z):
        if ((curr % 2) == 1) and ((curr % 3) == 0):
            print(curr)
        curr = curr + 1
```

9. Write a program that, given three numbers, a, b and c, finds the smallest, greatest and middle number. Assume none of the numbers are equal.

   Solution:

```
a = 1
b = 2
c = 3

if a > b:
    if a > c:
        if b > c:
            print("a largest, b middle, c smallest")
        else:
            print("a largest, c middle, b smallest")
    else:
        print("c largest, a middle, b smallest")
else:
    if a > c:
        print("b largest, a middle, c smallest")
    else:
        if b > c:
            print("b largest, c middle, a smallest")
        else:
            print("c largest, b middle, a smallest")
```

10. Write a program that does something interesting. This could be helping to solve a math problem, a short game or anything you want to code. Share your code on Piazza!

    Solution: Check Piazza for some solutions!