homework-02

July 8, 2022

1 Homework 2 - Conditionals, Predicates, & Quantifiers

Instructions: Complete the exercises below.

1. First, make a copy of this Colab Notebook to your GitHub account as a GitHub Gist (File -> Save a copy as a GitHub Gist). This will allow you to save any changes you make.
2. Now enter your answers to each question in the places shown below.
3. To submit your work, turn in the URL for your Colab notebook. Make sure the URL has the word `gist` in it. It should look something like this: `https://colab.research.google.com/gist/yourgithubusername/somerandomnumbersandletters/home`

Note: Sample solutions are provided in order to help you **learn**. The purpose of these exercises is not to simply turn in the answers, but to take the time and effort to learn and understand the material. You are encouraged to work the problems first, then when you are satisfied that you have the correct answer, look at the sample solution and compare it with yours.

1.0.1 Exercise 2.1

Let p be the proposition "I studied." and let q be the proposition "I got an A on the test."

Express the following propositions as English sentences.

1. $p \rightarrow q$
2. $\neg p \vee \neg q$
3. $\neg p \rightarrow (p \vee q)$
4. $\neg p \rightarrow \neg q$

Your Answer:

1.0.2 Exercise 2.2

True or false? Briefly explain your reasoning for each. 1. If $2 + 2 = 4$, then pigs can fly. 2. If $2 + 7 = 5$, then Elvis is alive. 3. If pigs can fly, then dogs can't fly. 4. If dogs have four legs, then ostriches have two legs. 5. $2 + 1 = 3$ if and only if $1 + 2 = 3$. 6. $1 + 2 = 3$ if and only if $3 + 1 = 6$. 7. $1 + 3 = 2$ if and only if the earth is flat. 8. $1 < 2$ if and only if $2 < 3$.

Your Answer:

1.0.3 Exercise 2.3

Let p be the proposition "You applied for college admission." and let q be the proposition "You were accepted."

Express the following English sentences as symbolic compound propositions using the logical conditional (or biconditional) connective.

1. Your applying for college admission is necessary for your being accepted.
2. Your being accepted is a sufficient condition for your applying for college admission.
3. Your applying for college admission is necessary and sufficient for your being accepted.

Your Answer:

1.0.4 Exercise 2.4

Write Python functions to implement the conditional operator "$\rightarrow$" (implies) and the biconditional operator "$\leftrightarrow$" (if and only if, frequently written as "iff").

In other words, complete these functions:

[ ]:
```
def implies(p, q):
    """Implements the conditional operator ->
    """
    pass # replace with your code


def iff(p, q):
    """Implements the biconditional operator <-->
    """
    pass # replace with your code
```

Then use Python to print a truth table for $p \rightarrow q$ and $p \leftrightarrow q$ using your `implies` and `iff` functions to verify they are working correctly. The truth table should look something like this:

p	q	p->q	p<->q
T	T	T	T
T	F	F	F
F	T	T	F
F	F	T	T

Your Answer:

1.0.5 Exercise 2.5

The Boolean operators `and`, `or`, and `not` are built into Python. This is an exercise in implementing these operators arithmetically.

Your task is to implement the `not1`, `or1`, and `and1` functions below, **arithmetically** (i.e., do *not* the built-in Python `and`, `or`, `not`, or `if`). Use **only** the operators +, -, and *. The inputs to these functions will be zeros or ones only, with zero representing False and one representing True.

You'll know you got it right if the resulting test code returns True for all three functions.

[ ]:
```
def not1(p):
    pass # your code here

def and1(p, q):
    pass # your code here

def or1(p, q):
    pass # your code here


# Returns a bit string representing the final column
# in a truth table for the given function.
def test(function):
    # handle functions with 1 argument
    if function.__code__.co_argcount == 1:
        return ''.join([str(function(p)) for p in [0,1]])

    # otherwise assume 2 arugments
    return ''.join([str(function(p,q)) for p in [0,1] for q in [0,1]])


print(test(not1) == '10') # Test not1 function
print(test(and1) == '0001') # Test and1 function
print(test(or1) == '0111') # Test or1 function
```

1.0.6 Exercise 2.6

The domain of discourse for this exercise is the set of all people, and:
$P(x) =$ x is older than 21
$S(x) =$ x is a student

Express in good English the following statements: 1. $\exists x\ P(x)$. 2. $\forall x\ P(x)$. 3. $\exists x\ \neg P(x)$. 4. $\forall x\ \neg P(x)$. 5. $\exists x\ S(x)$. 6. $\forall x\ S(x)$. 7. $\neg\exists x\ S(x)$. 8. $\exists x\ \neg S(x)$. 9. $\neg\forall x\ \neg S(x)$. 10. $\forall x\ \neg S(x)$.

Your Answer:

1.0.7 Hints:

1. Don't use the word 'exists'.
2. Don't use the word 'all'.
3. Use the word 'someone'.
4. 'No one' sounds better than 'Everyone is not'.
5. 'Some' <something> 'is a' <something else>.

6. Start with ‘Everyone’.
7. Start with ‘There is’.
8. Start with ‘There is’
9. Start with ‘No all’.
10. Start with ‘No one’.

1.0.8 Exercise 2.7

The domain of discourse for this exercise is the set of all people, and:

$F(x) =$ x is a friend
$C(x) =$ x is cool
$S(x) =$ x is a student
$N(x) =$ x is from Nepal

Express in good English the following statements: 1. $\forall x\ (F(x) \rightarrow C(x))$ 2. $\exists x\ (F(x) \wedge C(x))$ 3. $\forall x\ (F(x) \wedge C(x))$ 4. $\exists x\ (F(x) \rightarrow C(x))$ 5. $\forall x\ (S(x) \rightarrow N(x))$ 6. $\exists x\ (S(x) \rightarrow N(x))$ 7. $\forall x\ (S(x) \wedge N(x))$ 8. $\exists x\ (S(x) \wedge N(x))$

Your Answer:

1.0.9 Hints

1. Start with ‘All’.
2. Start with ‘Some’.
3. Start with ‘Everyone’.
4. Start with ‘Some’.
5. Start with ‘All’.
6. Start with ‘Someone’ and use ‘if’ and ‘then’.
7. Start with ‘Everyone’.
8. ‘Some student ...’.

1.0.10 Exercise 2.8

Translate the following statements into logical expressions using predicates, quantifiers, and logical connectives:

1. Everyone’s a critic.
2. No one is perfect.
3. At least one of your friends is perfect.
4. All of your friends are critics.
5. Everyone is a critic or someone is your friend.
6. No one is a critic and everyone is your friend.
7. Some state has no neighboring states.
8. Whenever there is an error, at least one error message is displayed.
9. All the programs have been scanned, but at least one program has a virus.

Your Answer:

1.0.11 Hints

1. Use the predicates C (is a Critic), P (is Perfect), and F (is your Friend).
2. Remember to push the negation sign to the right of the quantifier (and change the flavor of the quantifier).
3. Remember which connective to use with the existential quantifier?
4. Remember which connective to use with the universal quantifier?
5. This looks like the disjunction of two quantified statements.
6. This looks like the conjunction of two quantified statements, one of them negated.
7. Use a unary *and* a binary predicate.
8. There are two different domains at play here.
9. Use three predicates with programs as their domain.

1.0.12 Exercise 2.9

1. Write a function `is_even(x)` that returns `True` if `x` is even.

   Test this function with several different values for `x` to make sure it is working the way you expect.

   Notice that `is_even(x)` is a predicate. In other words, it is a function that returns `True` if the input `x` has the property defined by the predicate.

2. Now write a function called `forall` that takes two parameters, `p` and `s`, where `p` is a predicate and `s` is a list of numbers. `forall` should return `True` if every item in `s` has the property `p`.

   Test the `forall` function by passing in your `is_even` function as the first argument and the list `s = [2, 4, 6, 8, 10]` as the second argument. The result should be `True` because every number in `s` is even.

   Then test the `forall` function by passing in your `is_even` function with the list `s = [1, 2, 3, 4, 5]`. The result should be `False`, because not *every* number in `s` is even.

[ ]:
```
# Define a predicate that returns True if x is even
def is_even(x):
    pass # your code here


# Returns True if p is True for all integers in s
def forall(p, s):
    pass # your code here
```

You'll know you have correctly implemented `forall` if the following code block returns True for each test.

[ ]:
```
# Test forall and is_even with a list containing only even numbers
s = [2, 4, 6, 8, 10]
print(forall(is_even, s) == True)


# Test forall and is_even with a list containing some numbers that are not even
```

```
s = [1, 2, 3, 4, 5]
print(forall(is_even, s) == False)
```

1.0.13 Exercise 2.10

Write a function called `forsome` that takes a predicate `p` and a list `s`. `forsome` should return `True` if *at least one* number in the list `s` has the property `p`.

Test the `forsome` function by passing in your `is_even` function as the first argument and the list `s = [1, 3, 5, 7, 9]` as the second argument. The result should be `False`.

Then test the `forsome` function by passing in your `is_even` function with the list `s = [1,2,3,4,5,6]`. The result should be `True`, because at least one number in `s` is even.

[ ]:
```
def forsome(p, s):
    pass # your code here
```

You'll know you have correctly implemented `forsome` if the following code block returns True for each test.

[ ]:
```
# Test forsome and is_even with a list containing only odd numbers
s = [1, 3, 5, 7, 9]
print(forsome(is_even, s) == False)


# Test forsome with a list containing some numbers that are even
s = [1, 2, 3, 4, 5]
print(forsome(is_even, s) == True)
```

1.0.14 Exercise 2.11

In this problem, the domain is the integers `[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]`.

Determine the truth value of each of the following:

1. $\forall x$(`x is odd`)
2. $\exists x$(`x is odd`)
3. $\forall x$(`x is negative`)
4. $\exists x$(`x is negative`)
5. $\forall x(x^2 - x \neq 1)$
6. $\forall x(x^2 > 0)$
7. $\exists x(x^2 > 0)$
8. $\forall x(x \leq x + 1)$
9. $\exists x(x + x = 1)$
10. $\exists x(x + 2 = 1)$

Then use your `forall` and `forsome` functions to determine the truth value of each statement. You will need to write a predicate function for each one, similar to the `is_even` function you wrote previously. (You are encouraged to use *lambda* functions, but you may also use regular named functions)

Your Answer:

1.1 Sample Solutions

Note: Sample solutions are provided in order to help you **learn**. The purpose of these exercises is not to simply turn in the answers, but to take the time and effort to learn and understand the material. You are encouraged to work the problems first, then when you are satisfied that you have the correct answer, look at the sample solution and compare it with yours.

1.1.1 Sample Solution 2.1

1. If I studied then I got an A on the test.
2. I did not study or I did not get an A on the test.
3. If I didn't study then I studied or I got an A on the test.
4. If I did not study then I did not get an A on the test.

1.1.2 Sample Solution 2.2

1. This is false, because "2 + 2 = 4" is true but "pigs can fly" is false.
2. Both parts are false, so the conditional is true.
3. This is true, because pigs can't fly, and neither can dogs, so we have false $\rightarrow$ true.
4. We have true $\rightarrow$ true, so this is true.
5. As we have true $\leftrightarrow$ true, this is true.
6. As we have true $\leftrightarrow$ false, this is false.
7. As we have false $\leftrightarrow$ false, this is true.
8. Again we have true $\rightarrow$ true, so true.

1.1.3 Sample Solution 2.3

1. $p \leftarrow q$ (or $q \rightarrow p$)
2. $q \rightarrow p$
3. $p \leftrightarrow q$

1.1.4 Sample Solution 2.4

[ ]:
```
def implies(p, q):
    return not p or q

# Another way:
def implies(p, q):
    return q if p else True
```

```python
# Another way:
def implies(p, q):
    if p:
        return q
    else:
        return True

def iff(p, q):
    return implies(p, q) and implies(q, p)


operators = {
    'implies': ' ',
    'iff': ' ',
}

# Print a truth table
def print_truth_table(operation):
    print(f"p       q       p {operators[operation.__name__]} q")
    for p in [True, False]:
        for q in [True, False]:
            print(f'{p!s:<8}{q!s:<8}{operation(p,q)!s:<8}')
    print()

print_truth_table(implies)
print_truth_table(iff)
```

1.1.5 Sample Solution 2.5

[ ]:
```python
def not1(p):
    return 1 - p

def and1(p, q):
    return p * q

def or1(p, q):
    return (p + q) - (p * q)
```

1.1.6 Sample Solution 2.6

1. Somebody is older than 21.
2. Everyone is older than 21.
3. Someone is not older than 21.
4. No one is older than 21.
5. Someone (or some person) is a student.
6. Everyone is a student.
7. There is no student.

8. There is someone who is not a student.
9. Not all people are non-students.
10. No one is a student.

1.1.7 Sample Solution 2.7

1. All friends are cool.
2. Someone is a friend and is cool.
3. Everyone is a friend and is cool.
4. Some friends are cool. (Better than 'For some person, if that person is a friend, then that person is cool.')
5. All students are from Nepal.
6. Someone, if he/she is a student then he/she is from Nepal. (Better than 'For some person, if that person is a student, then that student is from Nepal.)
7. Everyone is a student and is from Nepal.
8. Some student is from Nepal.

1.1.8 Sample Solution 2.8

1. $\forall x\ C(x)$
2. $\forall x\ \neg P(x)$
3. $\exists x\ (F(x) \wedge P(x))$
4. $\forall x\ (F(x) \rightarrow C(x))$
5. $\forall x\ C(x) \vee \exists x\ F(x)$
6. $\neg\exists x\ C(x) \wedge \forall x\ F(x)$
7. $\exists x\ State(x) \wedge (\forall y\ State(y) \rightarrow \neg ShareABorder(x, y))$
8. $\exists x\ is\_an\_Error(x) \rightarrow (\exists y\ is\_an\_error\_Message(y) \wedge is\_Displayed(y))$
9. $[\forall x\ (is\_a\_Program(x) \rightarrow has\_been\_Scanned(x))] \wedge [\exists x\ (is\_a\_Program(x) \wedge has\_a\_Virus(x))]$

1.1.9 Sample Solution 2.9

[ ]:
```
def is_even(x):
    return not(x % 2)

def forall(p, s):
    for x in s:
        if not p(x):
            return False
    return True
```

1.1.10 Sample Solution 2.10

[ ]:
```
def forsome(p, s):
    for x in s:
        if p(x):
            return True
    return False
```

1.1.11 Sample Solution 2.11

[ ]:
```
domain = list(range(-5, 6))

print(f'1. {forall(lambda x: x % 2, domain)}')
print(f'2. {forsome(lambda x: x % 2, domain)}')
print(f'3. {forall(lambda x: x < 0, domain)}')
print(f'4. {forsome(lambda x: x < 0, domain)}')
print(f'5. {forall(lambda x: x**2 - x != 1, domain)}')
print(f'6. {forall(lambda x: x^2 > 0, domain)}')
print(f'7. {forsome(lambda x: x^2 > 0, domain)}')
print(f'8. {forall(lambda x: x <= x + 1, domain)}')
print(f'9. {forsome(lambda x: x + x == 1, domain)}')
print(f'10. {forsome(lambda x: x + 2 == 1, domain)}')
```