Seatwork 2.1
Creating, Modules and Packages.
PSMDSRC103
BASAL, RAFFY

Activity 1. Exploring Built-in Functions.

The program systematically computes the truth value of a logical expression for all possible combinations of input values (truth values) for the given variables. The combination list is generated using binary equivalents, ensuring every possible value combination is covered. The binary equivalents refer to the representation of numbers in the binary numeral system, which uses only two symbols: 0 and 1. Binary equivalents are used to represent all possible combinations of truth values (True or False, represented as 1 or 0, respectively) for the variables in propositional logic. This shows how the logical expression behaves with different inputs, giving the user insight into the behavior of the propositional logic formula.

Activity 2. Using the open() function for File Handling

The problem of this code is the open file is not the same as the file name for the “text” defined in the file writer.

```
1 file = open("new.txt", 'r')
2 data = file.read()
3 print(data)
4 file.close()
```

Should be “ file = open("newfile.txt", 'r')”

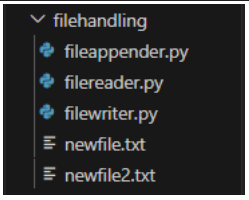
Appending newfile2.txt and opening the file, it will display what’s in the file. You can import files from the outside.

```
file = open("newfile2.txt", 'r')
data = file.read()
print(data)
file.close()
```

✓ 0.0s

```
and also by the programmer, of course.
```

New file were created along with the file handling folder:



Activity 3. User-defined Functions

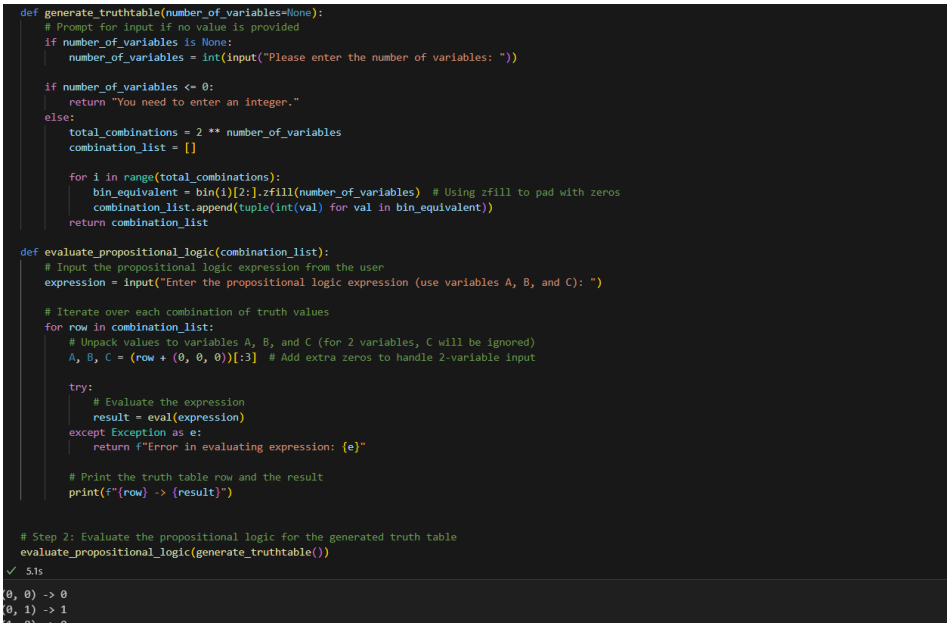
1. When the program generate_truthtable was run, it shows 8, (2**3=8), possible combination of the binary equivalents.
2. I modify the code to have an input by the user, but it displays same result if the user will input “NONE” or “0”.

```
def generate_truthtable(number_of_variables=None):
    # Prompt for input if no value is provided
    if number_of_variables is None:
        number_of_variables = int(input("Please enter the number of variables: "))

    if number_of_variables <= 0:
        return "You need to enter an integer."
    else:
        total_combinations = 2 ** number_of_variables
        combination_list = []

        for i in range(total_combinations):
            bin_equivalent = bin(i)[2:]
            while len(bin_equivalent) < number_of_variables:
                bin_equivalent = "0"+ bin_equivalent
            combination_list.append(tuple(int(val) for val in bin_equivalent))
        return combination_list
print(generate_truthtable())
```

✓ 2.6s

```
You need to enter an integer.
```

3. The evaluate_proportional_logic () is a code that the approach makes the code interactive and ensures that the user can customize the logic expression and the number of variables dynamically. This setup allows you to generate a truth table and evaluate a propositional logic expression using that table. It can handle both 2 variable and 3 variable logics.



```
def generate_truthtable(number_of_variables=None):
    # Prompt for input if no value is provided
    if number_of_variables is None:
        number_of_variables = int(input("Please enter the number of variables: "))

    if number_of_variables <= 0:
        return "You need to enter an integer."
    else:
        total_combinations = 2 ** number_of_variables
        combination_list = []

        for i in range(total_combinations):
            bin_equivalent = bin(i)[2:].zfill(number_of_variables)  # Using zfill to pad with zeros
            combination_list.append(tuple(int(val) for val in bin_equivalent))
        return combination_list

def evaluate_propositional_logic(combination_list):
    # Input the propositional logic expression from the user
    expression = input("Enter the propositional logic expression (use variables A, B, and C): ")

    # Iterate over each combination of truth values
    for row in combination_list:
        # Unpack values to variables A, B, and C (for 2 variables, C will be ignored)
        A, B, C = (row + (0, 0, 0))[:3]  # Add extra zeros to handle 2-variable input

        try:
            # Evaluate the expression
            result = eval(expression)
        except Exception as e:
            return f"Error in evaluating expression: {e}"

        # Print the truth table row and the result
        print(f"{row} -> {result}")


# Step 2: Evaluate the propositional logic for the generated truth table
evaluate_propositional_logic(generate_truthtable())
```

✓ 5.1s

```
(0, 0) -> 0
(0, 1) -> 1
```

The difference of generate_truthtable() or traditional and evaluate_proportional_logic() representation with the use of TUPLES.

Traditional Truth Table Format are much more Readable Format, it's in a standard format for truth tables used in textbooks and in teaching logic in discrete mathematics. It is also easier to read and interpret because it lays out all the combinations and their corresponding results in a tabular manner.

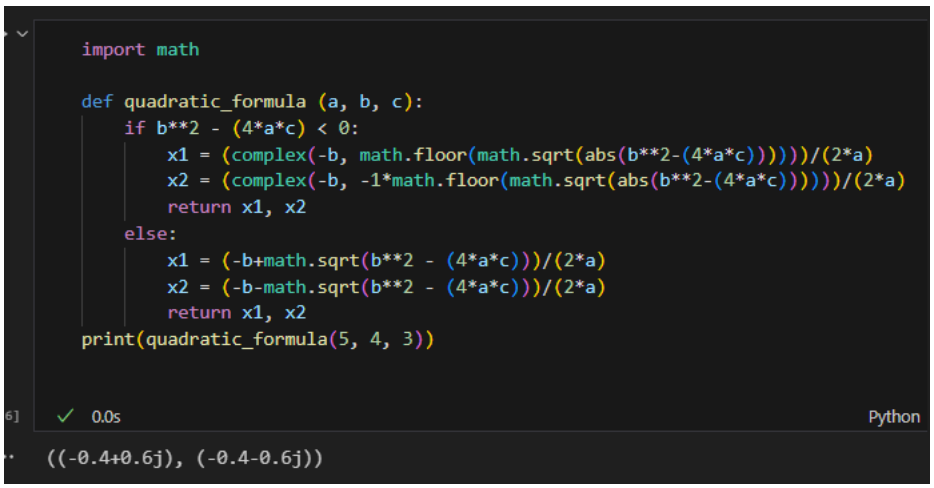
Tuple Format with Evaluation Results, this format is less compact and can be harder to scan for large numbers of variables. However, it clearly shows how each combination of truth values maps to the evaluated result. This kind of format is often generated by 3 or other programming languages, which print the result of each evaluation with a clear mapping from input (truth values) to output (result).

Both formats serve the same purpose—representing the results of propositional logic evaluations—but their structure and intended use cases are different.

Activity 4. Modules: Built-in Modules

MATH MODULES

1. In the first example in **mathmodule.py**, it is a quadratic formula that compute for the exponential value of the given coefficients. It resulted to a complex roots or the imaginary roots,



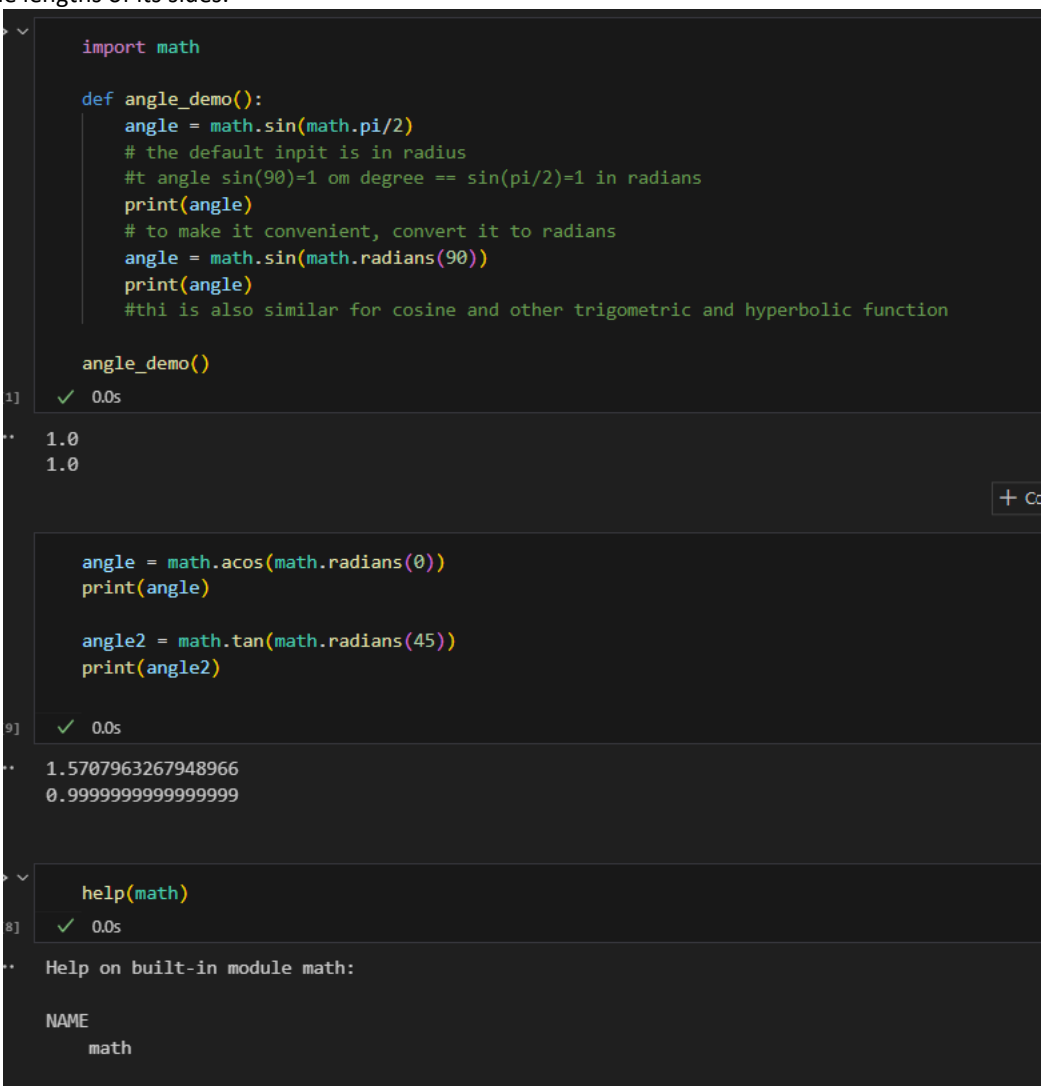
```
import math

def quadratic_formula (a, b, c):
    if b**2 - (4*a*c) < 0:
        x1 = (complex(-b, math.floor(math.sqrt(abs(b**2-(4*a*c))))))/(2*a)
        x2 = (complex(-b, -1*math.floor(math.sqrt(abs(b**2-(4*a*c))))))/(2*a)
        return x1, x2
    else:
        x1 = (-b+math.sqrt(b**2 - (4*a*c)))/(2*a)
        x2 = (-b-math.sqrt(b**2 - (4*a*c)))/(2*a)
        return x1, x2
print(quadratic_formula(5, 4, 3))
```

✓ 0.0s Python

```
((-0.4+0.6j), (-0.4-0.6j))
```

which is represented as complex numbers.

2. The second activity, this is a math function, a fundamental trigonometric function used in mathematics, particularly in geometry, physics, and engineering. These functions relate the angles of a right triangle to the lengths of its sides.



```
import math

def angle_demo():
    angle = math.sin(math.pi/2)
    # the default inpit is in radius
    #t angle sin(90)=1 om degree == sin(pi/2)=1 in radians
    print(angle)
    # to make it convenient, convert it to radians
    angle = math.sin(math.radians(90))
    print(angle)
    #thi is also similar for cosine and other trigometric and hyperbolic function

angle_demo()
```

```
[1] ✓ 0.0s
1.0
1.0
```

```
angle = math.acos(math.radians(0))
print(angle)

angle2 = math.tan(math.radians(45))
print(angle2)
```

```
[9] ✓ 0.0s
1.5707963267948966
0.9999999999999999
```

```
help(math)
```

```
[8] ✓ 0.0s
Help on built-in module math:

NAME
    math
```

TIME and DATETIME MODULE

Count down timer – it would display countdown based on the range you define. For this example, the define range is (10, 0, -1), meaning it start (10) and stops before it reaches (0), and the loop decrements (-1) every time until it ends. I added a pause of 1 second to delay the counting and will look like a countdown.

Current Time – this would display the current time today. It's define as time.strftime ("%I:%M %p").
time.strftime – is Pythons data or time modules, and must be a string format.
%I - 12hrs format, ":" -separator, %M – mins (0-59), %p – AM or PM
%H – 24hrs format, ":" -separator, %M – mins (0-59)

Current Date – the current date, define as ("%b %d %Y").
%b – the month
%d – days in the month
%Y - Year

```python
import time

def pause():
    for i in range(10, 0, -1):
        print(f"The program will end in {i}...")
        time.sleep(1)#the counting will commence after a second(1)

def current_time():
    t = time.strftime("%I:%M %p")  # Fixed typo
    return t

def current_date():
    d = time.strftime("%b %d %Y")
    return d

pause()
print(current_time())
print(current_date())
```

✓ 10.0s Python

```
The program will end in 10...
The program will end in 9...
The program will end in 8...
The program will end in 7...
The program will end in 6...
The program will end in 5...
The program will end in 4...
The program will end in 3...
The program will end in 2...
The program will end in 1...
11:44 PM
Sep 15 2024
```

USERS-DEFINED MODULES

The last activity, the dateandtime.py is used or imported in the main.py. You need to be specific with what you will going to import to avoid unwanted data. It created a new directory, name __pycache__, this automatically created by Python when Python source files are executed. It contains compiled bytecode files, which are versions of your Python scripts that have been translated into an intermediate format (bytecode) that the Python interpreter can run more efficiently.