Foundational Math 1

July 4, 2024

Learn Foundational Math 1 by Building an Equation Solver Each of these steps will lead you toward the Certification Project. First you have to copy the files and set them up in your Google Drive.

1 ↓ Do this first ↓

Make sure you are logged into your Google account, and copy this notebook to your own account. Click "File" (at the top of this page) and then click "Save a copy in Drive." The file will be in a folder called "Colab Notebooks" in your Google Drive.

#Directions - Click to expand the next step Click on the little triangle next to the word "Step" to do that step. Once you complete a step, click the triangle to expand the next step.

2 Step 0 - Acquire the testing library

Please run this code to get the library file from FreeCodeCamp. Each step will use this library to test your code. You do not need to edit anything; just run this code cell and wait a few seconds until it tells you to go on to the next step.

[1]:
```
# You may need to run this cell at the beginning of each new session

!pip install requests

# This will just take a few seconds

import requests

# Get the library from GitHub
url = 'https://raw.githubusercontent.com/freeCodeCamp/cdn/main/build/
↪math-cert-tests/math-code-test-a.py'
r = requests.get(url)

# Save the library in a local working directory
with open('math_code_test_a.py', 'w') as f:
    f.write(r.text)

# Now you can import the library
import math_code_test_a as test
```

```
# This will tell you if the code works
test.step19()
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests) (2024.6.2)

 Test passed. You can go on to the next step.
```

3 Step 1 - Add

To help you get familiar with Colab notebooks, you will start with the basics. Python uses `+`, `-`, `*`, and `/` for the four math operations: add, subtract, multiply, and divide. When you add two numbers, the result is the sum. Use addition within the `print` statement to get the sum of `a` and `b`. To run the code, you can hit "shift" and "enter" or you can click the run button (the triangle inside a circle).

[2]:
```
a=1
b=2

# Change the next line to print the sum of a and b
print(a+b)


# Only change code above this line
import math_code_test_a as test
test.step01(In[-1].split('# Only change code above this line')[0])
```

```
3

Code test passed
Go on to the next step
```

4 Step 2 - Subtract

When you subtract two numbers, the result is the difference. Use subtraction in the print statement to get the difference between `c` and `d`. Remember to use "shift" and "enter" to run the code.

[3]:
```
c = 7
d = 3

# Change the next line to print the positive difference between c and d
print(c-d)


# Only change code above this line
import math_code_test_a as test
test.step02(In[-1].split('# Only change code above this line')[0])
```

```
4

Code test passed
Go on to the next step
```

5 Step 3 - Multiply

When you multiply numbers, the result is the product. Use multiplication within the print statement to get the product of `e` and `f`:

[4]:
```
e = 2
f = 4

# Change the next line to print the product of e and f
print(e*f)


# Only change code above this line
import math_code_test_a as test
test.step03(In[-1].split('# Only change code above this line')[0])
```

```
8

Code test passed
Go on to the next step
```

6 Step 4 - Divide

When you divide two numbers, the result is the quotient. Use division within the print statement to get the quotient of `g` and `h`:

[5]:
```
g = 8
h = 4

# Change the next line
print(g/h)
```

```



# Only change code above this line
import math_code_test_a as test
test.step04(In[-1].split('# Only change code above this line')[0])
```

```
2.0

Code test passed
Go on to the next step
```

7 Step 5 - Cast Input

User input comes in as a string, so you need to cast it as an integer or a float before doing any math. The code below asks for input and uses `int()` to cast it as an integer. Follow the model and cast the second variable as an integer. Then run the code an test it. (Remember to hit "enter" after you type each integer in the box.)

[6]:
```
strA = input('Enter a positive integer: ')
intA = int(strA)

strB = input('Enter another positive integer: ')

# Change the next line but keep the variable name:
intB = int(strB)


print(intA+intB)


# Only change code above this line
import math_code_test_a as test
test.step05(In[-1].split('# Only change code above this line')[0])
```

```
Enter a positive integer: 4
Enter another positive integer: 3
7

Code test passed
Go on to the next step
```

#Step 6 - Input and cast on the same line

You can prompt for input and cast that input on the same line. Notice the nested functions in the first line of code. Follow that model to prompt for input and cast that input as an integer on the same line.

[7]:
```python
intA = int(input('Enter an integer: '))

# Change the next line but keep the variable name:
intB = int(input('Enter an integer: '))


print(intA+intB)


# Only change code above this line
import math_code_test_a as test
test.step06(In[-1].split('# Only change code above this line')[0])
```

```
Enter an integer: 3
Enter an integer: 6
9

Code test passed
Go on to the next step
```

#Step 7 - Float numbers

A float number allows decimal places. When prompting for a number as input, casting that as a float is usually the best choice. Follow the model below to prompt for input and cast that input as a float on the same line.

[8]:
```python
a = float(input('Enter a number: '))

# Change the next line but keep the variable name:
b = float(input('Enter a number: '))


print(a/b)


# Only change code above this line
import math_code_test_a as test
test.step07(In[-1].split('# Only change code above this line')[0])
```

```
Enter a number: 3
Enter a number: -9
-0.3333333333333333

Code test passed
Go on to the next step
```

8 Step 8 - Order of Operations

You may have heard of the order of operations and the acronym PEMDAS, which reminds you of the correct order. This means that you do what is in Parentheses first, then simplify Exponents. You then do all of the Multiplication and Division together, as long as you work from left to right and simplify them in order. The same is true of Addition and Subtraction: work from left to right and simplify the one the comes up next. Python knows the order of operations. In the following code, Python will calculate the actual_answer correctly. Notice the use of `**` to indicate an exponent. Do the arithmetic in your head (no writing code) and change the `your_answer` variable, then run the code to see if your_answer matches the actual_answer.

[9]:
```
actual_answer = (1+4*2-14/2)**3

# Put your answer on the following line:
your_answer = 8

print('Actual answer is ', actual_answer)
print('Your answer is ', your_answer)


# Only change code above this line
import math_code_test_a as test
test.step08(your_answer)
```

```
Actual answer is  8.0
Your answer is  8

Code test passed
Go on to the next step
```

9 Step 9 - Remainder and Modulus

A remainder is what is left over when you try to divide two numbers and it doesn't divide evenly. The remainder of 10 / 4 is 2 because 4 goes into 10 two whole times, with 2 left over. The modulus (%) operator will output the remainder, so `10 % 4` will return 2. Use the modulus operator to find the remainder of a divided by b:

[10]:
```
a = 14
b = 6

# Change this line
print(a%b)


# Only change code above this line
import math_code_test_a as test
test.step09(In[-1].split('# Only change code above this line')[0])
```

```
2

Code test passed
Go on to the next step
```

10 Step 10 - Modulus and Factors

Use an `if` statement with the modulus operator to find out if one number is a factor of another. For example, to see if 5 is a factor of 20, you can test `if 20 % 5 == 0`. If there's no remainder, the second number is a factor of the first. Remember that Python comparisons use `==` to test values. Remember that the `if` statement ends in a colon (`:`) and the resulting block is indented four spaces. Finish the code below to print "true" if `test_factor` is a factor of `number` and print "false" if it is not.

[11]:
```python
number = int(input('Enter an integer: '))
test_factor = int(input('Enter an integer to see if it's a factor: '))

# Change the next line to test the factor:
if number % test_factor == 0:
    print('true')
else:
    print('false')


# Only change code above this line
import math_code_test_a as test
test.step10(In[-1].split('# Only change code above this line')[0])
```

```
Enter an integer: 12
Enter an integer to see if it's a factor: 6
true

Code test passed
Go on to the next step
```

11 Step 11 - Finding Factors

Now you will find all of the factors of a number. This code has a loop with a variable, `test_factor`, that iterates through a defined range. Remember that the first line defining the loop ends in a colon (:) and each line in the loop requires a four-space indent. Change the `if` statement to find all the factors of `number`.

[12]:
```python
number = int(input('Enter an integer: '))

# Only change the if statement:
for test_factor in range(1, number+1):
    if number % test_factor == 0:
        print(test_factor)
```

```
# Only change code above this line
import math_code_test_a as test
test.step11(In[-1].split('# Only change code above this line')[0])
```

```
Enter an integer: 45
1
3
5
9
15
45

Code test passed
Go on to the next step
```

12 Step 12 - Prime Numbers

A prime number is a number whose only factors are 1 and itself. The number 5 is prime because its only factors are 1 and 5, but the 6 is not prime because it has 1, 2, 3, and 6 as factors. Any number that is not a prime is a composite. For each iteration in the loop, `test_number` will be a possible factor. Change the `if` statement so that the code prints “composite” if `number` is not prime.

[13]:
```
number = int(input("Enter a positive integer: "))

prime_or_comp = "prime"

for test_number in range(2,number):
    # Change the if statement to test one factor here:
    if number % test_number == 0:
        prime_or_comp = "composite"

print(prime_or_comp)


# Only change code above this line
import math_code_test_a as test
test.step12(In[-1].split('# Only change code above this line')[0])
```

```
Enter a positive integer: 211
prime

Code test passed
Go on to the next step
```

13 Step 13 - Reciprocals

A *reciprocal* is a number “flipped.” The reciprocal of $\frac{2}{3}$ is $\frac{3}{2}$ and the reciprocal of 5 is $\frac{1}{5}$ because whole numbers have denominators of 1. You can multiply a number by its reciprocal to get 1, so 5 $* \frac{1}{5} = 1$ and $\frac{2}{3} * \frac{3}{2} = 1$. To get the reciprocal of a number, take 1 divided by that number. Trying to get the reciprocal of zero will lead to a “divide by zero” error. Use a print statement to output the reciprocal of `n` as a decimal.

[14]:
```
n = float(input('Enter a number: '))

# Write your code here
print(1/n)



# Only change code above this line
import math_code_test_a as test
test.step13(In[-1].split('# Only change code above this line')[0])
```

```
Enter a number: 4
0.25

Code test passed
Go on to the next step
```

14 Step 14 - Splitting input

The code below asks for two integers, separated by a comma, then splits the input at the comma. Notice the input remains a string, then the `split()` function creates an array with two elements. Finish the following code to cast the two variables `a` and `b` as `float` numbers, then divide the two numbers and print the result.

[15]:
```
nums = input('Enter two numbers, separated by a comma: ')
sp = nums.split(",")

# Use the next line as a model:
a = float(sp[0])

# Change the next line to cast the number as a float:
b = float(sp[1])

# Change the print statement:
print(a/b)



# Only change code above this line
import math_code_test_a as test
test.step14(In[-1].split('# Only change code above this line')[0])
```

```
Enter two numbers, separated by a comma: 5, 2
2.5

Code test passed
Go on to the next step
```

15 Step 15 - Square Numbers

One factor multiplied by itself will produce a square number, so a number raised to an exponent of 2 is that number squared (like calculating the area of a square). Python uses `**` to indicate exponents. Complete the code to print the square of the input.

[16]:
```python
n = float(input('Enter a number to square: '))

# Change this line of code:
print(n**2)


# Only change code above this line
import math_code_test_a as test
test.step15(In[-1].split('# Only change code above this line')[0])
```

```
Enter a number to square: 7
49.0

Code test passed
Go on to the next step
```

16 Step 16 - Square Root Function

You can find the square root of a number with the `sqrt()` function. To use this function, you need to import the math library. This library enables you to use many functions, as you will see in later steps. To get the square root of x, you would write `math.sqrt(x)`. Complete the code to print the square root of a number.

[17]:
```python
import math

n = float(input('Enter a number to find the square root: '))

# Change the next line of code:
print(math.sqrt(n))


# Only change code above this line
import math_code_test_a as test
test.step16(In[-1].split('# Only change code above this line')[0])
```

```
Enter a number to find the square root: 361
19.0

Code test passed
Go on to the next step
```

17 Step 17 - Floor Function

The`floor()` function drops any decimals and sometimes is called the integer part of a number. Complete the code to print the floor of a number. Notice you `import math` and use `math.floor(n)`

[18]:
```python
import math

n = float(input('Enter a number with decimal places: '))

# Change the next line of code:
print(math.floor(n))



# Only change code above this line
import math_code_test_a as test
test.step17(In[-1].split('# Only change code above this line')[0])
```

```
Enter a number with decimal places: 3.1415926535
3

Code test passed
Go on to the next step
```

18 Step 18 - Finding Square Factors

This step will combine a few things you have already done. Remember that a square number is an integer that is the result of multiplying another integer by itself. Just as you created a loop to find factors of an integer, here you will find the greatest factor that is a perfect square. For example, 2 is a factor of 16, but 2 is not a square number, while 4 is a factor and it is a square number, but it is not the greatest square factor. The greatest square factor of 16 is 16. The greatest square factor of 32 is 16. Complete `if` statement in the loop to find the greatest square factor of a number.

[19]:
```python
import math

n = int(input('Enter an integer to find the greatest square factor: '))

max_factor = 1
upper_limit = math.floor(math.sqrt(n)) + 1

# Change one line in this loop:
for maybe_factor in range(1,upper_limit):
    if n % (maybe_factor**2) == 0:
        max_factor = maybe_factor

# Keep this print statement:
print(max_factor**2)


# Only change code above this line
import math_code_test_a as test
test.step18(In[-1].split('# Only change code above this line')[0])
```

```
Enter an integer to find the greatest square factor: 240
16

Code test passed
Go on to the next step
```

19 Step 19 - Dividing out Factors

Building upon your code from the previous step, this code will divide out the greatest square factor of a number. You don't need to change anything; just run the code below a few times, inputting different numbers each time.

[20]:
```
import math

n = int(input('Enter an integer to factor: '))
upper_limit = math.floor(math.sqrt(n)) + 1
square_root = 1
max_factor = 1
other_factor = 1

# Notice what the loop is doing here
for maybe_factor in range(1, upper_limit):
    # Check for square factors
    if n % (maybe_factor**2) == 0:
        # Find the greatest square factor
        max_factor = maybe_factor**2

# Divide out the greatest square factor
other_factor = n/max_factor

# Display the results
print("", n, " = ", max_factor, " * ", other_factor)


# Only change code above this line
import math_code_test_a as test
```

```
test.step19()
```

```
Enter an integer to factor: 88
 88  =  4  *  22.0

 Test passed. You can go on to the next step.
```

20 Step 20 - Factoring Square Roots

The last four steps prepared you for this. To factor a square root, you want to divide out any perfect square factors. For example: $\sqrt{12} = \sqrt{4*3} = 2\sqrt{3}$ Because 4 is a square number, the square root of 4 is now outside the radical. You will import `sympy` and `symbols` to use the radical ($\sqrt{x}$) in the output. Use the code from the previous step (without changing much). Your goal is to ask for a number and output the factored square root. The radical formatting (using sympy and symbols) is already done for you.

[21]:
```
import math
import sympy
from sympy import symbols

n = int(input('Without the radical, enter a square root to factor: '))

# Use these variables
upper_limit = math.floor(math.sqrt(n)) + 1
max_factor = 1
other_factor = 1
square_root = 1

# Notice what the loop is doing here
for maybe_factor in range(1, upper_limit):
    if n % (maybe_factor**2) == 0:
        max_factor = maybe_factor**2

# Divide out the greatest square factor
other_factor = n/max_factor

# Output - keep this:
square_root = int(math.sqrt(max_factor))
other_factor = int(other_factor)
output = square_root*sympy.sqrt(other_factor)


# Only change code above this line
import math_code_test_a as test
test.step20()
output
```

```
Without the radical, enter a square root to factor: 112

 Test passed. This is this your factored square root:
```

[21]: $4\sqrt{7}$

21 Step 21 - Rounding

If you only want a certain number of decimal places, use the `round()` function. This takes two arguments: the number to round and the number of decimal places, so `round(2.468, 2)` will return `2.47`. To round a large number instead of a decimal number, make the second argument negative, so `round(2345, -3)` will return `2000`. Finish the code below so that it prints the first number rounded to the nearest million (six zeros) and the second number rounded to 3 decimal places.

[22]:
```python
a = 14588132
b = 0.006538298336

# Write your code here
print(round(a, -6))
print(round(b, 3))


# Only change code above this line
import math_code_test_a as test
test.step21(In[-1].split('# Only change code above this line')[0])
```

```
15000000
0.007

Code test passed
Go on to the next step
```

#Step 22 - Fractions, Decimals, Percents

To convert a decimal number to a fraction, let x = the number of decimal places. The basic fraction is the number (without the decimal point) over 10^x. Example: $0.2 = \frac{2}{10}$ and $0.34 = \frac{34}{100}$ and 0.567 $= \frac{567}{1000}$. In some cases, you may be able to reduce that fraction. Because "percent" means "out of 100" the percent refers to the first two decimal places. Complete the code to ask for a decimal input, then print the fraction and the percent. Hint: The `exponent` variable gives you the number of decimal places.

[23]:
```python
import math

digits = input("Enter a decimal number to convert: ")
exponent = int(len(digits))-1
n = float(digits)
```

```
# Change the values of these three variables
numerator = int(n*10**(exponent))
denominator = 10**(exponent)
percent = n*100

# Output - keep this
print("The decimal is ", n)
print("The fraction is ", numerator, "/", denominator)
print("The percent is ", percent, " %")


# Only change code above this line
import math_code_test_a as test
test.step22(n,numerator,denominator,percent,exponent)
```

```
Enter a decimal number to convert: 0.125
The decimal is  0.125
The fraction is  1250 / 10000
The percent is  12.5  %

Code test passed
Go on to the next step
```

#Step 23 - Defining a Function

To execute a block of code with one command, define a function with the `def` command and the name of the function. Notice everything in the function is indented 4 spaces. Run the following code to see an example. Then change the function name to `fun()` and also change the name where you call the function. Run the code again.

[24]:
```
# Define a function
def fun():
    print("This is in the function")

# Other code not in the function
print("This is outside the function")

# Call the function
fun()

print("Back outside the function")

# Only change code above this line
import math_code_test_a as test
test.step23(In[-1].split('# Only change code above this line')[0])
```

```
This is outside the function
This is in the function
Back outside the function
```

```
Code test passed
Go on to the next step
```

22 Step 24 - Function with Input

A function can take input (called an "argument") and do something with that input. Use `def` to define the function, and include a variable in the parentheses to represent the argument. Indent everything that is a part of the function. When calling the function, pass the argument to it in the parentheses. Run the following code to see this example. Then change the `input()` variable name to `nombre` and also change that variable name in the argument when you call the function. Run the code again.

[25]:
```python
# Define a function
def greeting(nombre):
    print("Hello ", nombre)

nombre = input("What is your name? \n")

# Call the function
greeting(nombre)

# Only change code above this line
import math_code_test_a as test
test.step24(In[-1].split('# Only change code above this line')[0])
```

```
What is your name?
William
Hello  William

Code test passed
You can go on to the next step
```

#Step 25 - Function with Two Inputs

To pass more than one argument to a function, separate the arguments with commas. Run the code to see the example, then add a third argument to the function and run it again. The `third` variable is already in the code. Change three lines of code to use that variable: (1) the argument when you call the function, (2) the argument when you define the function, (3) the `sum` line within in the function definition.

[27]:
```python
# Define function
def add(a,b,c):
  # Use c for the third variable
  sum = a+b+c
  print("The sum is ", sum)

first = float(input("Enter a number: \n"))
second= float(input("Enter another number: \n"))
```

```
third = float(input("Enter another number: \n"))

# Call the function
add(first,second,third)

# Only change code above this line
import math_code_test_a as test
test.step25(In[-1].split('# Only change code above this line')[0])
```

```
Enter a number:
14
Enter another number:
3
Enter another number:
5
The sum is  22.0

Code test passed
You can go on to the next step
```

#Step 26 - Function with Return Value

Instead of including a `print()` statement within the function, the function can `return` a value right where you call it. To make the function return a value, use the `return` statement. Run the following code to see an example, then change the `return` statement to multiply by 3 instead of 2 and run the code again.

[28]:
```
# define the function
def multiplied(number):
    return number*3

a = float(input("Enter a number: \n"))
print("Your number multiplied = ", multiplied(a))


# Only change code above this line
import math_code_test_a as test
test.step26(In[-1].split('# Only change code above this line')[0])
```

```
Enter a number:
4
Your number multiplied =  12.0

Code test passed
You can go on to the next step
```

23 Step 27 - Solving for x

In Algebra, `X` often refers to the unknown number in an equation. To find the value of `x` we use algebra rules to get to `x =` [some number]. SymPy is a Python library to work with symbolic math. The following code works to solve an equation set equal to zero. Run the code and remember to use Python syntax to enter an equation (with “x” as the variable) and see the solution.

[29]:
```
import sympy
from sympy import symbols
from sympy.solvers import solve

x = symbols('x')

eq = input('Enter an equation to solve for x: 0 = ')
print(len(solve(eq,x)))
print("x = ", solve(eq,x)[0])


# Only change code above this line
import math_code_test_a as test
test.step27(In[-1].split('# Only change code above this line')[0])
```

```
Enter an equation to solve for x: 0 = 3*x - 9
1
x =  3
If you didn't get a syntax error, you are ready for the project
```

24 Step 28 - Make your own Functions

Building upon what you did in previous steps, define a different function for each of the following:

Add, subtract, multiply, divide

Detect prime numbers

Generate prime factors of a number

Simplify square roots

Solve for a variable

Each function should prompt the user with a question, take input, and output the answer.

[30]:
```
import math
import sympy
from sympy import symbols
from sympy.solvers import solve

# CALCULATOR FUNCTIONS

def add():
    a = input("Enter a first summand: ")
    b = input("Enter a second summand: ")
    if is_number(a) and is_number(b):
        result = float(a) + float(b)
        print(f"The sum of your numbers is {result}.")
    else:
        for val in (a, b):
            if not is_number(val):
                print(f"ERROR: {val} is not a valid number!")

def subtract():
    a = input("Enter a minuend: ")
    b = input("Enter a subtrahend: ")
    if is_number(a) and is_number(b):
        result = float(a) - float(b)
        print(f"The difference of your numbers is {result}.")
    else:
        for val in (a, b):
            if not is_number(val):
                print(f"ERROR: {val} is not a valid number!")

def multiply():
    a = input("Enter a first factor: ")
    b = input("Enter a second factor: ")
    if is_number(a) and is_number(b):
        result = float(a) * float(b)
        print(f"The product of your numbers is {result}.")
    else:
        for val in (a, b):
            if not is_number(val):
                print(f"ERROR: {val} is not a valid number!")

def divide():
    a = input("Enter a dividend: ")
    b = input("Enter a divisor: ")
    if is_number(a) and is_number(b):
        result = float(a) / float(b)
        print(f"The quotient of your numbers is {result}.")
    else:
        for val in (a, b):
            if not is_number(val):
                print(f"ERROR: {val} is not a valid number!")

def prime_factors():
    a = input("Enter a whole number: ")
    try:
        prime_factors = ", ".join(map(str, get_prime_factors(int(a))))
        print(f"The prime factors of {a} are: {prime_factors}.")
    except ValueError:
        print(f"ERROR: {a} is not a valid number!")

def simplify_sqrt():
    a = input("Without the radical, enter a square root to factor: ")
    try:
        a = int(a)
        upper_limit = math.floor(math.sqrt(a)) + 1
        max_factor = 1
        other_factor = 1
        square_root = 1
        for nr in range(1, upper_limit):
            if a % (nr**2) == 0:
                max_factor = nr**2
        other_factor = a/nr

        square_root = int(math.sqrt(max_factor))
        other_factor = int(other_factor)
        output = square_root*sympy.sqrt(other_factor)
    except ValueError:
        print(f"{a} is not a valid input!")

def solve_for_x():
    x = symbols("x")
    eq = input("Enter an equation to solve for x: 0 = ")
    res = solve(eq, x)
    print("x = ", res)

def get_binary_representation():
    a = input("Enter a whole number: ")
    try:
        print(f"The binary representation of your number is {int(a):08b}.")
    except ValueError:
        print(f"ERROR: {a} is not a valid number!")


# HELPER FUNCTIONS

def get_prime_factors(val):
    i = 2
    factors = []
    while i * i <= val:
        if val % i:
            i += 1
        else:
            val //= i
            factors.append(i)
    if val > 1:
        factors.append(val)
    return factors

def is_number(val):
    try:
        float(val)
        return True
    except ValueError:
        return False


#prime_factors()
#solve_for_x()

# This step does not have test
```

25 Step 29 - Create a Menu

Use print statements to create a menu that displays a numbered list of options. Then prompt for user input to choose an option. Use an `if` statement to print a different message for each option in the menu.

[31]:
```
# Write your code here
print("*** WELCOME TO THIS LITTLE CALCULATOR PROGRAM ***")
print("You can select one of the following options in order to carry out the␣
 ↪specified mathematical operation: ")
print("  1. Add two numbers\n  2. Subtract two numbers\n  3. Multiply two␣
 ↪numbers\n  4. Divide two numbers")
print("  5. Get the prime factors of a (whole) number\n  6. Simplify the square␣
 ↪root of a number\n  7. Solve for variable")
print("  8. Get the binary representation of a (whole) number")

user_option = input("Please choose one of these options: ")
match user_option:
    case "1":
        add()
    case "2":
        subtract()
    case "3":
        multiply()
    case "4":
        divide()
    case "5":
        prime_factors()
    case "6":
        simplify_sqrt()
    case "7":
        solve_for_x()
    case "8":
        get_binary_representation()
    case _:
        print("--> You did not select any of the specified options!")

# This step does not have test
```

```
*** WELCOME TO THIS LITTLE CALCULATOR PROGRAM ***
You can select one of the following options in order to carry out the specified
mathematical operation:
  1. Add two numbers
  2. Subtract two numbers
  3. Multiply two numbers
  4. Divide two numbers
  5. Get the prime factors of a (whole) number
  6. Simplify the square root of a number
  7. Solve for variable
  8. Get the binary representation of a (whole) number
Please choose one of these options: 8
Enter a whole number: 4
The binary representation of your number is 00000100.
```

26 Step 30 - Certification Project 1

Now put it all together to build a multi-function calculator. Use the menu and the functions you created in the previous steps. Define one more function of your own. Create the menu so that the user input will run a function.

[32]:
```
# Write your code here
import math
import sympy
from sympy import symbols
from sympy.solvers import solve

###############################################
# CALCULATOR FUNCTIONS

def add():
    a = input("Enter a first summand: ")
    b = input("Enter a second summand: ")
    if is_number(a) and is_number(b):
        result = float(a) + float(b)
        if result == int(result):
            result = int(result)
        print(f"The sum of your numbers is {result}.")
    else:
        for val in (a, b):
            if not is_number(val):
                print(f"ERROR: {val} is not a valid number!")

def subtract():
    a = input("Enter a minuend: ")
    b = input("Enter a subtrahend: ")
    if is_number(a) and is_number(b):
        result = float(a) - float(b)
        if result == int(result):
            result = int(result)
        print(f"The difference of your numbers is {result}.")
    else:
        for val in (a, b):
            if not is_number(val):
                print(f"ERROR: {val} is not a valid number!")

def multiply():
    a = input("Enter a first factor: ")
    b = input("Enter a second factor: ")
    if is_number(a) and is_number(b):
        result = float(a) * float(b)
        if result == int(result):
            result = int(result)
        print(f"The product of your numbers is {result}.")
    else:
        for val in (a, b):
            if not is_number(val):
                print(f"ERROR: {val} is not a valid number!")

def divide():
    a = input("Enter a dividend: ")
    b = input("Enter a divisor: ")
    if is_number(a) and is_number(b):
        result = float(a) / float(b)
        if result == int(result):
            result = int(result)
        print(f"The quotient of your numbers is {result}.")
    else:
        for val in (a, b):
            if not is_number(val):
                print(f"ERROR: {val} is not a valid number!")

def prime_factors():
    a = input("Enter a whole number: ")
    try:
        prime_factors = ", ".join(map(str, get_prime_factors(int(a))))
        print(f"The prime factors of {a} are: {prime_factors}.")
    except ValueError:
        print(f"ERROR: {a} is not a valid number!")

def simplify_sqrt():
    a = input("Without the radical, enter a square root to factor: ")
    try:
        a = int(a)
        upper_limit = math.floor(math.sqrt(a)) + 1
        max_factor = 1
        other_factor = 1
        square_root = 1
        for nr in range(1, upper_limit):
            if a % (nr**2) == 0:
                max_factor = nr**2
        other_factor = a/max_factor

        square_root = int(math.sqrt(max_factor))
        other_factor = int(other_factor)
        print(f"The simplified square root of your number is {square_root *␣
 ↪sympy.sqrt(other_factor)}.")
    except ValueError:
        print(f"ERROR: {a} is not a valid input!")

def solve_for_x():
    x = symbols("x")
    eq = input("Enter an equation to solve for x: 0 = ")
    res = solve(eq, x)
    print("x = ", res)

def get_binary_representation():
    a = input("Enter a whole number: ")
    try:
        print(f"The binary representation of your number is {int(a):08b}.")
    except ValueError:
        print(f"ERROR: {a} is not a valid number!")


################################################
# HELPER FUNCTIONS

def get_prime_factors(val):
    i = 2
    factors = []
    while i * i <= val:
        if val % i:
            i += 1
        else:
            val //= i
            factors.append(i)
    if val > 1:
        factors.append(val)
    return factors

def is_number(val):
    try:
        float(val)
        return True
    except ValueError:
        return False


# MENU
print("*** WELCOME TO THIS LITTLE CALCULATOR PROGRAM ***")
print("You can select one of the following options in order to carry out the␣
 ↪specified mathematical operation: ")
print("  1. Add two numbers\n  2. Subtract two numbers\n  3. Multiply two␣
 ↪numbers\n  4. Divide two numbers")
print("  5. Get the prime factors of a (whole) number\n  6. Simplify the square␣
 ↪root of a number\n  7. Solve for variable")
print("  8. Get the binary representation of a (whole) number")

user_option = input("Please choose one of these options: ")
match user_option:
    case "1":
        add()
    case "2":
        subtract()
    case "3":
        multiply()
    case "4":
        divide()
    case "5":
        prime_factors()
    case "6":
        simplify_sqrt()
    case "7":
        solve_for_x()
    case "8":
        get_binary_representation()
    case _:
```

```python
        print("--> You did not select any of the specified options!")
```

```
*** WELCOME TO THIS LITTLE CALCULATOR PROGRAM ***
You can select one of the following options in order to carry out the specified
mathematical operation:
  1. Add two numbers
  2. Subtract two numbers
  3. Multiply two numbers
  4. Divide two numbers
  5. Get the prime factors of a (whole) number
  6. Simplify the square root of a number
  7. Solve for variable
  8. Get the binary representation of a (whole) number
Please choose one of these options: 5
Enter a whole number: 210
The prime factors of 210 are: 2, 3, 5, 7.
```

[ ]:
```

```