# Assignment 03

#### 1.Create a function to perform basic arithmetic operations that includes addition, subtraction, multiplication and division on a string number (e.g. "12 + 24" or "23 - 21" or "12 // 12" or "12 \* 21").

**Answer:**

Here, we have 1 followed by a space, operator followed by another space and 2. For the challenge, we are going to have only two numbers between 1 valid operator. The return value should be a number.  
eval() is not allowed. In case of division, whenever the second number equals "0" return -1.  
For example: "15 // 0" ➞ -1

**Examples:**

arithmetic\_operation("12 + 12") ➞ 24 // 12 + 12 = 24  
arithmetic\_operation("12 - 12") ➞ 24 // 12 - 12 = 0  
arithmetic\_operation("12 \* 12") ➞ 144 // 12 \* 12 = 144  
arithmetic\_operation("12 // 0") ➞ -1 // 12 / 0 = -1

import operator  
  
ops = {'+':operator.add,'-':operator.sub,'\*':operator.mul,'//':operator.floordiv}  
  
**def** arithmetic\_operation(in\_string):  
 in\_list = in\_string.split(" ")  
 output = ops[in\_list[1]](int(in\_list[0]),int(in\_list[2])) **if** int(in\_list[2]) != 0 **else** -1  
 print(f'arithmetic\_operation({in\_list[0]} {in\_list[1]} {in\_list[2]}) ➞ {output}')  
   
arithmetic\_operation("12 + 12")  
arithmetic\_operation("12 - 12")  
arithmetic\_operation("12 \* 12")  
arithmetic\_operation("12 // 0")

arithmetic\_operation(12 + 12) ➞ 24  
arithmetic\_operation(12 - 12) ➞ 0  
arithmetic\_operation(12 \* 12) ➞ 144  
arithmetic\_operation(12 // 0) ➞ -1

#### 2.Write a function that takes the coordinates of three points in the form of a 2d array and returns the perimeter of the triangle. The given points are the vertices of a triangle on a two-dimensional plane.

**Answer:**

**Examples:**  
perimeter( [ [15, 7], [5, 22], [11, 1] ] ) ➞ 47.08  
perimeter( [ [0, 0], [0, 1], [1, 0] ] ) ➞ 3.42  
perimeter( [ [-10, -10], [10, 10 ], [-10, 10] ] ) ➞ 68.28

import math  
  
**def** distance(a,b):  
 **return** math.sqrt(pow((b[1]-a[1]),2)+pow((b[0]-a[0]),2))  
  
**def** perimeter(in\_array):  
 perimeter = []  
 **for** ele **in** range(len(in\_array)):  
 **if** ele == len(in\_array)-1:  
 perimeter.append(distance(in\_array[ele],in\_array[0]))  
 **else**:  
 perimeter.append(distance(in\_array[ele],in\_array[ele+1]))  
 print(f'perimeter({in\_array}) ➞ {sum(perimeter):.2f}')  
   
perimeter([[15,7],[5,22],[11,1]])  
perimeter([[0,0],[0,1],[1,0]])  
perimeter([[-10,-10],[10,10],[-10,10]])

perimeter([[15, 7], [5, 22], [11, 1]]) ➞ 47.08  
perimeter([[0, 0], [0, 1], [1, 0]]) ➞ 3.41  
perimeter([[-10, -10], [10, 10], [-10, 10]]) ➞ 68.28

#### 3.A city skyline can be represented as a 2-D list with 1s representing buildings. In the example below, the height of the tallest building is 4 (second-most right column).

**Answer:**

[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 1, 0, 1, 0], [0, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1]]  
Create a function that takes a skyline (2-D list of 0's and 1's) and returns the height of the tallest skyscraper.  
**Examples:**

tallest\_skyscraper([[0, 0, 0, 0],[0, 1, 0, 0],[0, 1, 1, 0],[1, 1, 1, 1]]) ➞ 3  
tallest\_skyscraper([[0, 1, 0, 0],[0, 1, 0, 0],[0, 1, 1, 0],[1, 1, 1, 1]]) ➞ 4  
tallest\_skyscraper([[0, 0, 0, 0],[0, 0, 0, 0],[1, 1, 1, 0],[1, 1, 1, 1]]) ➞ 2

def tallest\_skyscraper(in\_list):  
 out\_list = []  
 for num in range(len(in\_list)):  
 count = 0  
 for ele in range(len(in\_list[num])):  
 count += in\_list[ele][num]  
 out\_list.append(count)  
 print(f'tallest\_skyscraper({in\_list}) ➞ {max(out\_list)}')  
   
tallest\_skyscraper([[0, 0, 0, 0],[0, 1, 0, 0],[0, 1, 1, 0],[1, 1, 1, 1]])  
tallest\_skyscraper([[0, 1, 0, 0],[0, 1, 0, 0],[0, 1, 1, 0],[1, 1, 1, 1]])  
tallest\_skyscraper([[0, 0, 0, 0],[0, 0, 0, 0],[1, 1, 1, 0],[1, 1, 1, 1]])

tallest\_skyscraper([[0, 0, 0, 0], [0, 1, 0, 0], [0, 1, 1, 0], [1, 1, 1, 1]]) ➞ 3  
tallest\_skyscraper([[0, 1, 0, 0], [0, 1, 0, 0], [0, 1, 1, 0], [1, 1, 1, 1]]) ➞ 4  
tallest\_skyscraper([[0, 0, 0, 0], [0, 0, 0, 0], [1, 1, 1, 0], [1, 1, 1, 1]]) ➞ 2

#### 4.A financial institution provides professional services to banks and claims charges from the customers based on the number of man-days provided. Internally, it has set a scheme to motivate and reward staff to meet and exceed targeted billable utilization and revenues by paying a bonus for each day claimed from customers in excess of a threshold target.

**Answer:**

This quarterly scheme is calculated with a threshold target of 32 days per quarter, and the incentive payment for each billable day in excess of such threshold target is shown as follows:

Days Bonus 0 to 32 days Zero 33 to 40 days SGD$325 per billable day 41 to 48 days SGD$550 per billable day Greater than 48 days SGD$600 per billable day

Please note that incentive payment is calculated progressively. As an example, if an employee reached total billable days of 45 in a quarter, his/her incentive payment is computed as follows:  
32\*0 + 8\*325 + 5\*550 = 5350  
Write a function to read the billable days of an employee and return the bonus he/she has obtained in that quarter.

#### Examples:

bonus(15) ➞ 0  
bonus(37) ➞ 1625  
bonus(50) ➞ 8200

def bonus(int\_num):  
 if int\_num > 48:  
 output = 0+(8\*325)+(8\*550)+((int\_num-48)\*600)  
 elif int\_num < 48 and int\_num >= 41:  
 output = 0+(8\*325)+((int\_num-41+1)\*550)  
 elif int\_num >33 and int\_num <= 40:  
 output = 0+((int\_num-33+1)\*325)  
 else:  
 output = 0  
 print(f'bonus({int\_num}) ➞ {output}')  
   
bonus(15)  
bonus(44)  
bonus(37)  
bonus(50)  
bonus(60)

bonus(15) ➞ 0  
bonus(44) ➞ 4800  
bonus(37) ➞ 1625  
bonus(50) ➞ 8200  
bonus(60) ➞ 14200

#### 5.A number is said to be Disarium if the sum of its digits raised to their respective positions is the number itself.

**Answer:**

Create a function that determines whether a number is a Disarium or not.  
**Examples:**

is\_disarium(75) ➞ False  
# 7^1 + 5^2 = 7 + 25 = 32  
is\_disarium(135) ➞ True  
# 1^1 + 3^2 + 5^3 = 1 + 9 + 125 = 135  
is\_disarium(544) ➞ False  
is\_disarium(518) ➞ True  
is\_disarium(466) ➞ False  
is\_disarium(8) ➞ True

**def** is\_disarium(in\_num):  
 sum = 0  
 output = False  
 **for** ele **in** range(len(str(in\_num))):  
 sum += int(str(in\_num)[ele])\*\*(ele+1)  
 **if** in\_num == sum:  
 output=True  
 print(f'is\_disarium({in\_num}) ➞ {output}')  
   
is\_disarium(75)  
is\_disarium(135)  
is\_disarium(544)  
is\_disarium(518)  
is\_disarium(466)  
is\_disarium(8)

is\_disarium(75) ➞ False  
is\_disarium(135) ➞ True  
is\_disarium(544) ➞ False  
is\_disarium(518) ➞ True  
is\_disarium(466) ➞ False  
is\_disarium(8) ➞ True