# AI Lab 1

# Exercise 1.1

**Type each of the following expressions into python3. What value do each of the following Python expressions evaluate to? Is that value an integer or a floating point?**

1. 250

|  |
| --- |
| 250 |

1. 28 % 5

|  |
| --- |
| 3 |

1. 2.5e2

|  |
| --- |
| 250.0 |

1. 3e5

|  |
| --- |
| 300000.0 |

1. 3 \* 10\*\*5

|  |
| --- |
| 300000 |

1. 20 + 35 \* 2

|  |
| --- |
| 110 |

*Why is this different from (20 + 35) \* 2?* g. 2 / 3 \* 3

1. 2 // 3 \* 3

*Why is this different from 2 / 3 \* 3?*

1. 25 - 5 \* 2 - 9

*Is this different from ((25 - 5) \* 2) - 9* *and/or 25 - ((5 \* 2) - 9)? Why?* **Exercise 1.2**

Suppose we are making ice cream sundaes. We have four flavors of ice cream: vanilla, chocolate, strawberry, and pistacchio. And we have three sauces: caramel, butterscotch, and chocolate. How many different ice cream sundaes can we make? Define a function sundaes() to systematically print out every possible combination, one per line. For example, the first line should say "vanilla ice cream sundae with caramel sauce". You should create a list to hold each class of ingredient, and use nested for loops to iterate over these lists to generate the combinations. At the end, your function should return an integer giving the total number of combinations. To get you started here a few things you will need:

flavors = ["vanilla", "chocolate", "strawberry", "pistacchio"] sauces = ["caramel", "butterscotch", "chocolate"]

print(flavor + " ice cream sundae with " + sauce + " sauce")

**Solution:**

|  |
| --- |
| flavors = ["vanilla", "chocolate", "strawberry", "pistacchio"]  sauces = ["caramel", "butterscotch", "chocolate"]  for x in flavors:      for y in sauces:          print(x + " ice cream sundae with " + y + " sauce") |

**Output:**

|  |
| --- |
| vanilla ice cream sundae with caramel sauce  vanilla ice cream sundae with butterscotch sauce  vanilla ice cream sundae with chocolate sauce  chocolate ice cream sundae with caramel sauce  chocolate ice cream sundae with butterscotch sauce  chocolate ice cream sundae with chocolate sauce  strawberry ice cream sundae with caramel sauce  strawberry ice cream sundae with butterscotch sauce  strawberry ice cream sundae with chocolate sauce  pistacchio ice cream sundae with caramel sauce  pistacchio ice cream sundae with butterscotch sauce  pistacchio ice cream sundae with chocolate sauce |

# Exercise 1.3

Consider the following output as shown:

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

16 17 18 19 20 21

22 23 24 25 26 27 28

29 30 31 32 33 34 35 36

37 38 39 40 41 42 43 44 45

46 47 48 49 50 51 52 53 54 55

Complete the missing parts so that it creates the output above. Note that the columns of numbers do not need to line up perfectly. Run your program in using python3 to test your work. HINT: Row i has i columns. def triangle(): value = 1 row = 1 while row <= **\_\_\_\_\_**:

column = 1 while column <= **\_\_\_\_\_\_\_**: if column != **\_\_\_\_\_\_\_**:

print(value, ' ', sep = '', end = '') else: print(value) value = value + 1 column = column + 1 row = row + 1

**Solution:**

|  |
| --- |
| value = 1      row = 1      while row <= 10:          column = 1          while column <= row:              if column != row:                  print(value, ' ', sep='', end='')              else:                  print(value)              value = value + 1              column = column + 1          row = row + 1 |

**Output:**

|  |
| --- |
| 1  2 3  4 5 6  7 8 9 10  11 12 13 14 15  16 17 18 19 20 21  22 23 24 25 26 27 28  29 30 31 32 33 34 35 36  37 38 39 40 41 42 43 44 45  46 47 48 49 50 51 52 53 54 55 |

# Exercise 1.4

1. Write the following functions:
   * cube(n), which takes in a number and returns its cube. For example, cube(3) => 27.

|  |
| --- |
| def cube(x):      return x \*\* 3  print(cube(3)) |

* + factorial(n), which takes in a non-negative integer n and returns n!, which is the product of the integers from 1 to n. (0! = 1 by definition.)

|  |
| --- |
| def factorial(n):      if n == 0:          return 1      else:          return n \* factorial(n-1)          print(factorial(5)) |

* + count\_pattern(pattern lst), which counts the number of times a certain pattern of symbols appears in a list, including overlaps. So count\_pattern( ('a', 'b'), ('a','b', 'c', 'e', 'b', 'a', 'b', 'f')) should return 2, and count\_pattern(('a', 'b', 'a'),

('g', 'a', 'b', 'a', 'b', 'a','b', 'a')) should return 3.

|  |
| --- |
| def count\_pattern(pattern, lst):      count = 0      n = len(pattern)      for i in range(len(lst) - n + 1):          if lst[i:i+n] == pattern:              count += 1      return count  print(count\_pattern(('a', 'b'), ('a', 'b', 'c', 'e', 'b', 'a', 'b', 'f')))  print(count\_pattern(('a', 'b', 'a'), ('g', 'a', 'b', 'a', 'b', 'a', 'b', 'a'))) |

* + Write a python program to print the multiplication table for the given number?

|  |
| --- |
| num = int(input("Enter the number for multiplication table: "))  for i in range(1, 11):     print(num, "x", i, "=", num \* i) |

* + Write a python program to implement Simple Calculator program? (+, -, / ,\*)

|  |
| --- |
| def add(num1, num2):      return num1 + num2  def subtract(num1, num2):      return num1 - num2  def multiply(num1, num2):      return num1 \* num2  def divide(num1, num2):      return num1 / num2  print("Select operation.")  print("1. Add")  print("2. Subtract")  print("3. Multiply")  print("4. Divide")  choice = input("Enter choice: ")  num1 = float(input("Enter first number: "))  num2 = float(input("Enter second number: "))  if choice == '1':      print(num1, "+", num2, "=", add(num1,num2))  elif choice == '2':      print(num1, "-", num2, "=", subtract(num1,num2))  elif choice == '3':      print(num1, "\*", num2, "=", multiply(num1,num2))  elif choice == '4':      print(num1, "/", num2, "=", divide(num1,num2))  else:      print("Invalid input") |

* + Write a python program to sort the sentence in alphabetical order?

|  |
| --- |
| sentence = input("Enter a sentence: ")  words = sentence.split()  words.sort()  sorted\_sentence = ' '.join(words)  print("Sorted sentence: ", sorted\_sentence) |
| Output:  Enter a sentence: a d c b  Sorted sentence: a b c d |

1. Write a Python class to convert an integer to a roman numeral.

|  |
| --- |
| class RomanNumeralConverter:      def \_\_init\_\_(self):          self.roman\_numerals = {              1000: "M",              900: "CM",              500: "D",              400: "CD",              100: "C",              90: "XC",              50: "L",              40: "XL",              10: "X",              9: "IX",              5: "V",              4: "IV",              1: "I"          }        def convert\_to\_roman(self, num):          roman\_numeral = ""          for value, symbol in self.roman\_numerals.items():              while num >= value:                  roman\_numeral += symbol                  num -= value          return roman\_numeral  converter = RomanNumeralConverter()  print(converter.convert\_to\_roman(1994)) |
| Output:  MCMXCIV |

1. Write a Python class to find a pair of elements (indices of the two numbers) from a

given array whose sum equals a specific target number. Input: numbers=

[10,20,10,40,50,60,70], target=50 Output: 3, 4

|  |
| --- |
| class PairFinder:      def \_\_init\_\_(self, numbers):          self.numbers = numbers        def find\_pair(self, target):          seen = {}          for i, num in enumerate(self.numbers):              complement = target - num              if complement in seen:                  return seen[complement], i              seen[num] = i          return None  numbers = [10, 20, 10, 40, 50, 60, 70]  target = 50  finder = PairFinder(numbers)  pair = finder.find\_pair(target)  if pair:      print(pair)  # Output: (2, 4)  else:      print("No pair found.") |

1. Write a Python class to find the three elements that sum to zero from a set of n real numbers. Input array : [-25, -10, -7, -3, 2, 4, 8, 10] Output : [[-10, 2, 8], [-7, -3, 10]].

|  |
| --- |
| class TripletFinder:      def \_\_init\_\_(self, nums):          self.nums = nums        def find\_triplets(self):          self.nums.sort()          triplets = []          for i in range(len(self.nums) - 2):              if i > 0 and self.nums[i] == self.nums[i - 1]:                  continue              left = i + 1              right = len(self.nums) - 1              while left < right:                  total = self.nums[i] + self.nums[left] + self.nums[right]                  if total < 0:                      left += 1                  elif total > 0:                      right -= 1                  else:                      triplets.append([self.nums[i], self.nums[left], self.nums[right]])                      while left < right and self.nums[left] == self.nums[left + 1]:                          left += 1                      while left < right and self.nums[right] == self.nums[right - 1]:                          right -= 1                      left += 1                      right -= 1          return triplets  nums = [-25, -10, -7, -3, 2, 4, 8, 10]  finder = TripletFinder(nums)  triplets = finder.find\_triplets()  print(triplets)  # Output: [[-10, 2, 8], [-7, -3, 10]] |

1. Write a Python class to reverse a string word by word. Input string : 'hello .py' Expected Output : '.py hello'

|  |
| --- |
| class StringReverser:      def \_\_init\_\_(self, s):          self.s = s        def reverse\_words(self):          words = self.s.split()          words.reverse()          return ' '.join(words)  s = 'hello .py'  reverser = StringReverser(s)  reversed\_string = reverser.reverse\_words()  print(reversed\_string) |

1. Count the numbers of characters in the string
   * 1. Read the string.
     2. Count the characters
     3. Display the result

|  |
| --- |
| string = input("Enter a string: ")  count = len(string)  print("The string contains", count, "characters.") |
| Output:  Enter a string: artificial  The string contains 10 characters. |

1. Addition of two square matrices.
   * 1. Create a lists to read matrix elements
     2. Read the elements of to matrices add the elements
     3. Store the result in third matrix.
     4. Repeat steps 2 and 3 till the addition of all elements

|  |
| --- |
| n = 3  matrix1 = [[0] \* n for \_ in range(n)]  matrix2 = [[0] \* n for \_ in range(n)]  result = [[0] \* n for \_ in range(n)]  print('Enter the elements of the first matrix:')  for i in range(n):      for j in range(n):          matrix1[i][j] = int(input(f'Element ({i},{j}): '))  print('Enter the elements of the second matrix:')  for i in range(n):      for j in range(n):          matrix2[i][j] = int(input(f'Element ({i},{j}): '))  for i in range(n):      for j in range(n):          result[i][j] = matrix1[i][j] + matrix2[i][j]  print('The sum of the matrices is:')  for i in range(n):      for j in range(n):          print(result[i][j], end=' ')      print() |
| Output:  Enter the elements of the first matrix:  Element (0,0): 1  Element (0,1): 2  Element (0,2): 3  Element (1,0): 4  Element (1,1): 5  Element (1,2): 6  Element (2,0): 7  Element (2,1): 8  Element (2,2): 9  Enter the elements of the second matrix:  Element (0,0): 9  Element (0,1): 8  Element (0,2): 7  Element (1,0): 6  Element (1,1): 5  Element (1,2): 4  Element (2,0): 3  Element (2,1): 2  Element (2,2): 1  The sum of the matrices is:  10 10 10  10 10 10  10 10 10 |

1. Display the result Multiplication of two matrices
   * 1. Create a lists to read matrix elements
     2. Read the elements of two matrices, multiply the elements
     3. Store the result in third matrix.
     4. Repeat steps 2 and 3 till the multiplication of all elements
     5. Display the result.

|  |
| --- |
| n = 3  # Number of rows and columns in each matrix  matrix1 = [[0] \* n for \_ in range(n)]  matrix2 = [[0] \* n for \_ in range(n)]  result = [[0] \* n for \_ in range(n)]  print('Enter the elements of the first matrix:')  for i in range(n):      for j in range(n):          matrix1[i][j] = int(input(f'Element ({i},{j}): '))  print('Enter the elements of the second matrix:')  for i in range(n):      for j in range(n):          matrix2[i][j] = int(input(f'Element ({i},{j}): '))  for i in range(n):      for j in range(n):          for k in range(n):              result[i][j] += matrix1[i][k] \* matrix2[k][j]  print('The product of the matrices is:')  for i in range(n):      for j in range(n):          print(result[i][j], end=' ')      print() |
| Output:  Enter the elements of the first matrix:  Element (0,0): 4  Element (0,1): 8  Element (0,2): 6  Element (1,0): 5  Element (1,1): 6  Element (1,2): 4  Element (2,0): 5  Element (2,1): 5  Element (2,2): 5  Enter the elements of the second matrix:  Element (0,0): 8  Element (0,1): 8  Element (0,2): 6  Element (1,0): 1  Element (1,1): 2  Element (1,2): 3  Element (2,0): 6  Element (2,1): 5  Element (2,2): 2  The product of the matrices is:  76 78 60  70 72 56  75 75 55 |

1. Write a function called calculator. It should take the following parameters: two numbers, an arithmetic operation (which can be addition, subtraction, multiplication or division and is addition by default), and an output format (which can be integer or floating point, and is floating point by default). Division should be floating-point division. The function should perform the requested operation on the two input numbers, and return a result in the requested format (if the format is integer, the result should be rounded and not just truncated). Raise exceptions as appropriate if any of the parameters passed to the function are invalid.

|  |
| --- |
| def calculator(num1, num2, operation='addition', output\_format='float'):      if not isinstance(num1, (int, float)) or not isinstance(num2, (int, float)):          raise ValueError("num1 and num2 must be numbers")        if operation not in ['addition', 'subtraction', 'multiplication', 'division']:          raise ValueError("Invalid operation")        if output\_format not in ['integer', 'float']:          raise ValueError("Invalid output format")        if operation == 'addition':          result = num1 + num2      elif operation == 'subtraction':          result = num1 - num2      elif operation == 'multiplication':          result = num1 \* num2      else:          result = num1 / num2        if output\_format == 'integer':          result = round(result)        return result |

1. Create a class called Numbers, which has a single class attribute called MULTIPLIER, and a constructor which takes the parameters x and y (these should all be numbers).
   * 1. Write a method called add which returns the sum of the attributes x and y.
     2. Write a class method called multiply, which takes a single number parameter a and returns the product of a and MULTIPLIER.
     3. Write a static method called subtract, which takes two number parameters, b and c, and returns b - c.
     4. Write a method called value which returns a tuple containing the values of x and y. Make this method into a property, and write a setter and a deleter for manipulating the values of x and y.

|  |
| --- |
| class Numbers:      MULTIPLIER = 2        def \_\_init\_\_(self, x, y):          if not isinstance(x, (int, float)) or not isinstance(y, (int, float)):              raise ValueError("x and y must be numbers")          self.\_x = x          self.\_y = y        def add(self):          return self.\_x + self.\_y        @classmethod      def multiply(cls, a):          if not isinstance(a, (int, float)):              raise ValueError("a must be a number")          return a \* cls.MULTIPLIER        @staticmethod      def subtract(b, c):          if not isinstance(b, (int, float)) or not isinstance(c, (int, float)):              raise ValueError("b and c must be numbers")          return b - c        @property      def value(self):          return (self.\_x, self.\_y)        @value.setter      def value(self, xy\_tuple):          if not isinstance(xy\_tuple, tuple) or len(xy\_tuple) != 2 or not isinstance(xy\_tuple[0], (int, float)) or not isinstance(xy\_tuple[1], (int, float)):              raise ValueError("xy\_tuple must be a tuple of two numbers")          self.\_x = xy\_tuple[0]          self.\_y = xy\_tuple[1]        @value.deleter      def value(self):          del self.\_x          del self.\_y |