Assignment 1

**Short Answers**

Answer the following questions with complete sentences in your own words. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers. You are expected to be able to explain your answers in detail. If you are unable to understand or answer some of the questions asked or the contents you have found, please provide the sources such as the website link or book, and mark your question in red.

1. What are indexes, how can you use indexing?

Indexes are the particular positions of elements within an iterable data type like a string or array. You can use indexing to perform computations with those elements in the iterable.

1. What is slicing? Please give an example.

Slicing is the way to access parts of an iterable. In Python the syntax is [start:stop:step],

So [0:10:2] would access the elements from 0 to 9 with step count 2. If the sample string is ‘123456789’ then it would return ‘13579’

1. What are strings? What are different ways to initialize a string?

A string is a collection of characters. Strings in python are immutable so they cannot be changed after they are created. Strings can be initialized using ‘ or ‘’.

1. List some common functions for strings, and explain their functionalities.

Startswith()/endswith() : returns true if string starts or ends with the input

Index(): returns index of value in the string

find(): similar to index but outputs -1 if the value is not found, while index throws an error

lower()/upper():converts string to lowercase/uppercase

replace():replace a value(s) in the string

format():formats specific values in a string

1. Why do we use string format?

We use format to complete a string dynamically, so we can fill parts of a string with particular values

1. What is dynamic typing, why it is useful. Please do some online research about what statically-typing is.

Dynamic typing allows for type inference, which means you don’t have to specify types every time. Type checking happens at program runtime. Statically typed languages do type checking at compile time, which means you have to specify types for every variable, which takes a longer time but is less prone to bugs.

1. What is a floating point number?

Numbers with a decimal point

1. List some common number operations.

Addition,subtraction,multiplication,floor division,float division,exponential, modulus

1. What are booleans?

Data type that has either True or False. Positive integers and nonempty data structures are True while empty data structures would be False.

1. When the if-else statement ends?

It ends depending on the condition of the if else statement

1. What are loops? What are nested loops?

Loops are a way to iterate through elements of an iterable. Nested loops are loops within loops

1. What’s the difference between break and continue statement?

Break stops the loops entirely while continue goes to the next iteration of the loop.

1. What is a function, list all features for the function.

A function is a code that runs when it is called. It contains parameters, which are inputs to the function. When a function is called, arguments are given for the parameters. A function can have a return value or can just be used to perform a certain action.

In terms of OOP, functions are objects, and can be stored in data structures or passed onto other functions.

1. What is a local variable?

A local variable is a variable declared with a scope limited to a loop or function and cannot be used outside of the scope in which it was declared.

1. How do you get input from users?

You can use the input() function to receive keyboard input.

1. What are lists in Python, how to initialize them?

A list is a built in data type in Python that can store multiple elements in a single variable. Lists can are ordered, changeable, and can hold duplicate values.

1. Write down some common list functions and explain their functionalities.

len(): returns length of list

range():returns an iterator of integers based on conditions given

sum():adds all items in a list

min()/max():returns min/max value in a list

sorted(): returns a sorted list

list():converts another iterable to a list

1. Why do we write comments?

We write comments so that the code is easier for us or someone else working with the code to understand. Comments can also explain the thought process for the code, so in long programs comments can be used to help debug.

**Coding Practice**

Write algorithms in Python to Complete following questions. Please write your own answers. You are highly encouraged to present more than one way to answer the questions. Please follow the syntax, and you need to comment on each piece of code you write to clarify your purpose. Also you need to clearly explain your logic of the code for each question. Clearly state your assumptions if you have any. You may discuss with others about the questions, but please write your own code. If you are unable to answer the questions, you can write the question in pseudocode but you need to explain your algorithm clearly in detail.

1. Reverse a number

Given an positive integer, try to reverse the number

Example 1:

Input: n = 12345

Output: 54321

Example 2:

Input: n = 1

Output: 1

# cast the integer into a string, use slicing to reverse

# and cast it back into an int

n = 12345

print(int(str(n)[::-1]))

2. Given an integer x, return true if x is a palindrome integer. An integer is a palindrome when it reads the same backwards as forward. For example, 121 is a palindrome while 123 is not.

Example 1:

Input: x = 121

Output: true

Example 2:

Input: x = -121S

Output: false

Example 3:

Input: x = 10

Output: false

#Cast the integer into a string and return true if the string is equal to the reversed string

def isPalindrome(x):

    if str(x) == str(x)[::-1]:

        return True

3. Write an algorithm to determine if a number n is happy.

A happy number is a number defined by the following process:

* Starting with a positive integer, replace the number by the sum of the squares of its digits.
* Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.
* Those numbers for which this process ends in 1 are happy.

Example 1:

Input: n = 19

Output: true

Explanation:

1^2 + 9^2 = 82

8^2 + 2^2 = 68

6^2 + 8^2 = 100

1^2 + 0^2 + 0^2 = 1

Example 2:

Input: n = 2

output : false

# Q3

# finds sums of squares of digits according to the given prompt

def happyMath(n):

    total = 0

    while n > 0:

        value = n % 10

        n //= 10

        total += value\*\*2

    return total

# check if there is a cycle

def isHappy(n):

    cycle = set()

    while n != 1 and n not in cycle:

        cycle.add(n)

        n = happyMath(n)

    return n == 1

4.Given a list nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the list.

Example 1:

Input: nums = [3, 0, 1]

Output: 2

Explanation: n = 3 since there are 3 numbers, so all numbers are in the range of [0,3]. 2 is the missing number in the range since it does not appear in nums.

Example 2:

Input: nums = [0, 1]

Output: 2

Example 3:

Input: nums = [9, 6, 4, 2, 3, 5, 7, 0, 1]

# Using a set with length n, subtract the set nums to find the missing value(s)

# Sorted turns it back into a sorted list so you can index it

def missing(nums):

    return sorted(set(range(0,len(nums)+1))-set(nums))[0]

5. You are given a large integer represented as an integer list digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0’s.

Increment the large integer by one and return the result list of digits.

Example 1:

Input: digits = [1, 2, 3]

Output: [1, 2, 4]

Explanation: The array represents the integer 123. Incrementing by one gives 123 + 1 = 124. Thus, the result should be [1, 2, 4].

Example 2:

Input: digits = [4, 3, 2, 1]

Output: [4, 3, 2, 2]

Explanation: The array represents the integer 4321. Incrementing by one gives 4321 + 1 = 4322. Thus, the result should be [4, 3, 2, 2].

Example 3:

Input: digits = [9]

Output: [1, 0]

Explanation: The array represents the integer 9. Incrementing by one gives 9 + 1 = 10. Thus, the result should be [1, 0].

First way that came to mind:

Turn the list into a single string, cast as an int, do the addition, and change it back into a list

def addOne(digits):

    number = ''

    newDigits = []

    for digit in digits:

        number += str(digit)

    number = str(int(number) + 1)

    for n in number:

        newDigits.append(int(n))

    return newDigits

List comprehension way:

Better way to turn the digits into a single integer in one line using join and map. Use list comprehension to turn the new integer into a list of integers

def plusOne(digits):

    number = int("".join(map(str, digits)))

    add = str(number + 1)

    res = [int(x) for x in add]

    return  res

6. Given a non-empty array of integers nums, every element appears twice except for one. Find that single one.

You must implement a solution with linear runtime complexity and use only constant extra space.

Example 1:

Input: nums = [2,2,1]

Output: 1

Example 2:

Input: nums = [4,1,2,1,2]

Output: 4

Example 3:

Input: nums = [1]

Output: 1

The same number XOR’ed with itself returns 0, so if you XOR all the values, the remaining value is the only number without a duplicate

# XOR of the same number returns 0

def single(nums):

    value = 0

    for num in nums:

        value ^= num

    return value

7.Given an integer array nums, move all 0’s to the end of it while maintaining the relative order of the non-zero elements.

Example 1:

Input: nums = [0, 1, 0, 3, 12]

Output: [1, 3, 12, 0, 0]

Example 2:

Input: nums = [0]

Output: [0]

The first way uses two pointers at the start and the end of the list. If the first pointer sees a 0, that 0 is deleted and a new 0 is appended at the end of the list, and the end pointer is moved back 1 to ignore the 0.

The second way counts the number of zeroes in the list, and fills the array with nonzero values before appending the correct number of zeroes

def moveZero(nums):

    pointer = 0

    end = len(nums)

    while pointer < end:

        if nums[pointer] == 0:

            del nums[pointer]

            nums.append[0]

            end -= 1

        else:

            pointer += 1

    return nums

#list comprehension

def moveZero1(nums):

    zeroes = nums.count(0)

    nums[:] = [i for i in nums if i != 0]

    nums.extend([0]\*zeroes)

    return nums

nums = [0, 1, 0, 3, 12]

8. Write a function that reverses a string. The input string is given as a list of characters s.

You must do this by modifying the input array in-place

Example 1:

Input: s = [“h”, “e”, “l”, “l”, “o”]

Output: [“o”, “l”, “l”, “e”, “h”]

Example 2:

Input: s = [“H”, “a”, “n”, “n”, “a”, “h”]

Output: [“h”, “a”, “n”, “n”, “a”, “H”]

In place sort requires swapping, so you can’t use slicing, which creates a new array/string. There’s also the reverse() function built into python, but that makes the prompt too easy.

This solution uses two pointers at the start and end, swaps the values, and moves the pointers closer to each other.

def reverseString(s):

    start = 0

    end = len(s) - 1

    while start < end:

        temp = s[start]

        s[start] = s[end]

        s[end] = temp

        start += 1

        end -= 1

    return s

9. Given a string s and a rotation factor r, rotate every alphanumeric character according to the rotation factor.

Example 1:

Input: s = “A-b!c”, r = 2

Output: “C-d!e”

Explanation:

“A” rotate 2 factors is “C”

“-” is not an alphanumeric character, therefore, no changes

“b” rotate 2 factors is “d”

“!” is not an alphanumeric character, therefore, no changes

“c” rotate 2 factors is “e”

Therefore, output is “C-d!e”

Example 2:

Input: s = “39ZA”, r = 4

Output: “73DE”

Rotating requires considering 4 cases if you use ASCII characters. If it’s not alphanumeric, then keep the character in the string the same.

Uppercase letters have ASCII codes from 65-90, so if you want the rotation, you need to first find out what the letter would be if A-Z were 0-25 and then mod 26 to wrap around, before adding 65 to get the right ASCII value. You go through a similar process for lowercase letters and digits, which have ASCII values from 97-122 and 48-47, respectively.

Since a string is immutable, I created a new string and appended each of the rotated letters from the original string.

def rotate(s,r):

    new\_string = ''

    for i in range(len(s)):

        #keep the same if not alphanumeric

        if s[i].isalnum() == False:

            new\_string += s[i]

            continue

        #uppercase letters

        if ord(s[i]) >= 65 and ord(s[i]) <= 90:

            letter = chr((ord(s[i])+r-65)%26 + 65)

        #lowercase letters

        elif ord(s[i]) >= 97 and ord(s[i]) <= 122:

            letter = chr((ord(s[i])+r-97)%26 + 97)

        #numbers

        elif ord(s[i]) >= 48 and ord(s[i]) <= 57:

            letter = chr((ord(s[i])+r-48)%10 + 48)

        new\_string += letter

    return new\_string