PYTHON PROGRAMMING LAB EXERCISES

MODULE 1

SIMPLE

1. WAP to find area of a Circle.

2. WAP to find perimeter of a Rectangle.

3. WAP to find the Simple Interest and the total amount when the Principal, Rate of Interest and Time are entered by the user.

4. WAP to convert °C to °F and °F to °C.

5. WAP to find distance between two points.

SELECTION

1. Write a program to find greatest of three numbers

2. Write a program to find whether a given number is even or not.

3. Write a program to whether a given number is prime or not.

4. Write a program to find whether a given string or number is palindrome or not

5. Given 3 sides, WAP to find whether a given triangle is right angled or not.

SELECTION AND BOOLEAN

1. cigar party

When squirrels get together for a party, they like to have cigars. A squirrel party is successful when the number of cigars is between 40 and 60, inclusive. Unless it is the weekend, in which case there is no upper bound on the number of cigars. Return True if the party with the given values is successful, or False otherwise.

cigar_party(30, False) → False

cigar_party(50, False) → True

cigar_party(70, True) → True

2. caught speeding

You are driving a little too fast, and a police officer stops you. Write code to compute the result, encoded as an int value: 0=no ticket, 1=small ticket, 2=big ticket. If speed is 60 or less, the result is 0. If speed is between 61 and 80 inclusive, the result is 1. If speed is 81 or more, the result is 2. Unless it is your birthday -- on that day, your speed can be 5 higher in all cases.

-caught_speeding(60, False) → 0

caught_speeding(65, False) → 1

caught_speeding(65, True) → 0

3. love6

The number 6 is a truly great number. Given two int values, a and b, return True if either one is 6. Or if their sum or difference is 6. Note: the function abs(num) computes the absolute value of a number.

love6(6, 4) → True

love6(4, 5) → False

love6(1, 5) → True

4. near ten

Given a non-negative number "num", return True if num is within 2 of a multiple of 10. Note: (a % b) is the remainder of dividing a by b, so (7 % 5) is 2.

near_ten(12) → True

near_ten(17) → False

near_ten(19) → True

5. make chocolate

We want make a package of goal kilos of chocolate. We have small bars (1 kilo each) and big bars (5 kilos each). Return the number of small bars to use, assuming we always use big bars before small bars. Return -1 if it can't be done.

make_chocolate(4, 1, 9) → 4

make_chocolate(4, 1, 10) → -1

make_chocolate(4, 1, 7) → 2

ITERATION

1. Print table of 1 to 10 in a tabular format.

2. Print all Prime Numbers in an Interval

3. Write a Python program to guess a number between 1 and 9. User is prompted to enter a guess. If the user guesses wrong then the prompt appears again until the guess is correct, on successful guess, user will get a "Well guessed!" message, and the program will exit.

4. Write a Python program to construct the following pattern, using a nested for loop.

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

5. Write a Python program that accepts a word from the user and reverse it.

6. Do above program without loop.

7. Write a program to find LCM and GCD of two numbers.

9. Write a program to find sum of all even numbers upto 30 except 10 and 20.

8. Find the Factorial of a Number

FUNCTION

1. make_bricks

We want to make a row of bricks that is goal inches long. We have a number of small bricks (1 inch each) and big bricks (5 inches each). WAF **make_bricks(small, big, goal)** that returns True if it is possible to make the goal by choosing from the given bricks.

make_bricks(3, 1, 8) → True

make_bricks(3, 1, 9) → False

make_bricks(3, 2, 10) → True

2. lone_sum

Given 3 int values, a b c, WAF **lone_sum(a, b, c)** that returns their sum. However, if one of the values is the same as another of the values, it does not count towards the sum.

lone_sum(1, 2, 3) → 6

lone_sum(3, 2, 3) → 2

lone_sum(3, 3, 3) → 0

3. no_teen_sum

Given 3 int values, a b c, WAF **no_teen_sum(a, b, c)** that returns their sum. However, if any of the values is a teen i.e. in the range 13 to 19 inclusive, then that value counts as 0. Write a separate helper function **fix_teen(n)** that takes in an int value and returns that value fixed for the teen rule. In this way, you avoid repeating the teen code 3 times (i.e. "decomposition").

no_teen_sum(1, 2, 3) → 6

no_teen_sum(2, 13, 1) → 3

no_teen_sum(2, 1, 14) → 3

MODULE 2

STRING SLICING AND INDEXING

1. Extract the elements of a string that have even indices

2. Make a copy of the same string in reverse order

3. Print even numbers upto 100, without using any arithmetic or comparison operator

4. Extract the elements of a string that are positioned on prime indices

5.

Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!".

helloName("Bob") → "Hello Bob!"

helloName("Alice") → "Hello Alice!"

helloName("X") → "Hello X!"

6.

Given two strings, a and b, return the result of putting them together in the order abba, e.g. "Hi" and "Bye" returns "HiByeByeHi".

makeAbba("Hi", "Bye") → "HiByeByeHi"

makeAbba("Yo", "Alice") → "YoAliceAliceYo"

makeAbba("What", "Up") → "WhatUpUpWhat"

7. The web is built with HTML strings like "Yay" which draws Yay as italic text. In this example, the "i" tag makes and which surround the word "Yay". Given tag and word strings, create the HTML string with tags around the word, e.g. "Yay".

makeTags("i", "Yay") → "Yay"

makeTags("i", "Hello") → "Hello"

makeTags("cite", "Yay") → "<cite>Yay</cite>"

8.

Given a string, return a new string made of 3 copies of the last 2 chars of the original string. The string length will be at least 2.

extraEnd("Hello") → "lololo"

extraEnd("ab") → "ababab"

extraEnd("Hi") → "HiHiHi"

9.

Given a string, return a version without the first and last char, so "Hello" yields "ell". The string length will be at least 2.

withoutEnd("Hello") → "ell"

withoutEnd("java") → "av"

withoutEnd("coding") → "odin"

10.

Given a string, return a "rotated right" version where the last n chars are moved to the start. The string length will be at least 2.

right2("Hello",2) → "loHel"

right2("java",3) → "avaj"

right2("Hi",3) → "Hi"

11.

Given a string, return a new string where the first and last chars have been exchanged.

'code' → 'eodc'

'a' → 'a'

'ab' → 'ba'

STRING AND SELECTION

1.

Given a string, we'll say that the front is the first 3 chars of the string. If the string length is less than 3, the front is whatever is there. Return a new string which is 3 copies of the front.

'Java' → 'JavJavJav'

'Chocolate' → 'ChoChoCho'

'abc' → 'abcabcabc'

2.

Given a string, return a new string where "not " has been added to the front. However, if the string already begins with "not", return the string unchanged.

'candy' → 'not candy'

'x' → 'not x'

'not bad' → 'not bad'

STRING MANIPULATION WITH METHODS

1. Find the Frequency of Characters in a String
2. Find the Number of Vowels, Consonants, Digits and White space in a String
3. Reverse a Sentence by Recursion
4. Find the Length of a String
5. Copy a String
6. Remove all Characters in a String except alphabet
7. Sort Elements in Lexicographical Order (Dictionary Order)
8. Check if a given String is Palindrome
9. Find the Largest & Smallest Word in a String
10. Remove all Characters in Second String which are present in First String
11. Delete All Repeated Words in String
12. Find the Frequency of the Word ‘the’ in a given Sentence
13. Frequency of Substring in the given String
14. Print the Words Ending with Letter ‘s’
15. Print all the duplicates in the input string.
16. Divide a string in N equal parts
17. Remove “b” and “ac” from a given string

18. Check if a given String is Anagram

LIST

1. Create a program that will keep track of items for a shopping list. The program should keep asking for new items until nothing is entered (no input followed by enter/return key). The program should then display the full shopping list.
2. Write a program that will store the schedule for a given day for a particular TV station. The program should ask you for the name of the station and the day of the week before asking you for the name of each show and the start and stop times. Once the schedule is complete it should be displayed as a table.
3. WAPP to multiply two matrix supplied by user in row-major representation.
4. Convert a string into characters
5. Sort the list by the length of string elements
6. Sort the list in reverse order
7. Given a non-empty array, return true if there is a place to split the array so that the sum of the numbers on one side is equal to the sum of the numbers on the other side.
   canBalance([1, 1, 1, 2, 1]) → true
   canBalance([2, 1, 1, 2, 1]) → false
   canBalance([10, 10]) → true
8. Given two arrays of ints sorted in increasing order, outer and inner, return true if all of the numbers in inner appear in outer. The best solution makes only a single "linear" pass of both arrays, taking advantage of the fact that both arrays are already in sorted order.
   linearIn([1, 2, 4, 6], [2, 4]) → true
   linearIn([1, 2, 4, 6], [2, 3, 4]) → false
   linearIn([1, 2, 4, 4, 6], [2, 4]) → true
9. Given n>=0, create an array length n*n with the following pattern, shown here for n=3 :
   [[0, 0, 1], [0, 2, 1], [3, 2, 1]] (spaces added to show the 3 groups).
   squareUp(3) → [[0, 0, 1],[0, 2, 1], [3, 2, 1]]
   squareUp(2) → [[0, 1], [2, 1]]
   squareUp(4) → [[0, 0, 0, 1], [0, 0, 2, 1], [0, 3, 2, 1], [4, 3, 2, 1]]
10. We'll say that a "mirror" section in an array is a group of contiguous elements such that somewhere in the array, the same group appears in reverse order. For example, the largest mirror section in {1, 2, 3, 8, 9, 3, 2, 1} is length 3 (the {1, 2, 3} part). Return the size of the largest mirror section found in the given array.
    maxMirror([1, 2, 3, 8, 9, 3, 2, 1]) → 3
    maxMirror([1, 2, 1, 4]) → 3
    maxMirror([7, 1, 2, 9, 7, 2, 1]) → 2