Week-8, Activity Questions

Problem-0

Print the namaste pattern `_/\_` n times. The pattern for $n = 10$ is given below:

```
_/\__/\__/\__/\__/\__/\__/\__/\__/\__/\_
```

Answer

```
n = int(input())
print('_/\_' * n)
```

Problem-1

Consider the line $y = x$. Write a function named `placement` that accepts a point (p, q) in 2-D space as input and returns `above` if this point is above the line, `on` if this point is on the line, and `below` if this point is below the line.

Answer

```python
def placement(p, q):
    if p == q:
        return 'on'
    if p < q:
        return 'above'
    if p > q:
        return 'below'
```

Problem-2

Find all integer Pythagorean triplets (x, y, z), with $0 < x < y < z < 1000$. Store them as a list of tuples. How many such triplets are there? Are there any triplets that satisfy the following condition: $z - y = 1$?

Answer

Basic, inefficient solution

This takes ages.

```
triplets = [ ]
for x in range(1, 1000):
    for y in range(x + 1, 1000):
        for z in range(y + 1, 1000):
            if x ** 2 + y ** 2 == z ** 2:
                triplets.append((x, y, z))

count = 0
for x, y, z in triplets:
    if z - y == 1:
        count += 1
```

Alternate, efficient solution

This is super-fast. Check out the live session to know more. This is a student-driven solution. Special thanks to Kumar Chandan for this solution.

```
triplets = [ ]
for x in range(1, 1000):
    for y in range(x + 1, 1000):
        z_ = (x ** 2 + y ** 2) ** 0.5
        if z_.is_integer():
            z = int(z_)
            if y < z < 1000:
                triplets.append((x, y, z))
```

Problem-3

Write a function that computes the sum of the first n terms of the series given below:

$$2^0 + 2^1 + 2^2 + \cdots + 2^{n-1}$$

Give two different implementations of the same:

- iterative
- recursive

Answer

```python
# iterative
def geometric_i(n):
    S = 0
    for i in range(n):
        S = S + 2 ** i
    return S

# recursive
def geometric_r(n):
    if n == 1:
        return 1
    return geometric_r(n - 1) + 2 ** (n - 1)
```

Problem-4

Write a function named `is_undirected` that accepts an adjacency-matrix representation of a graph as input. It should return `True` if the underlying graph is undirected and `False` otherwise.

Answer

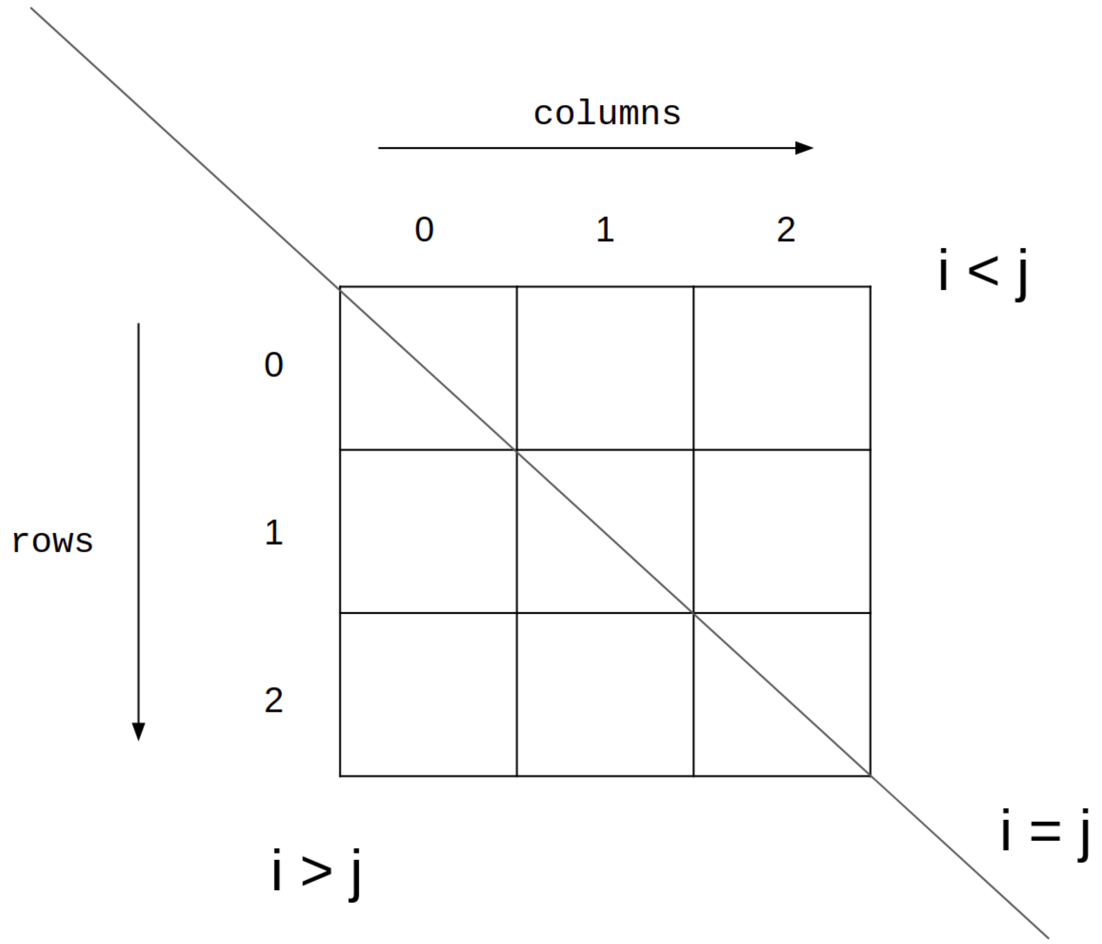
Let's assume that there are no edges from a node to itself. Also, we assume that the entries are going to be either 1 or 0.

Basic solution

Check for every cell in the matrix

```python
def is_undirected(mat):
    dim = len(mat)
    for i in range(dim):
        for j in range(dim):
            if mat[i][j] == 1 and mat[j][i] != 1:
                return False
    return True
```

More efficient solution



Check only for cells above the main diagonal

```python
def is_undirected(mat):
    dim = len(mat)
    for i in range(dim):
        for j in range(i + 1, dim):
            if mat[i][j] == 1 and mat[j][i] == 0:
                return False
            if mat[i][j] == 0 and mat[j][i] == 1:
                return False
    return True
```

We are essentially checking if the adjacency matrix A is symmetric: $A[i][j] = A[j][i]$, as it should be for an undirected graph.

Problem-5

Write a recursive function to multiply two positive integers a and b. You can only use $+$ and $-$ operators. You are not allowed to use the $*$ symbol anywhere in your code!

Credits: Cornell university

Answer

Multiplication is repeated addition.

Basic solution

```python
# Process this as follows:
# the number a should be added b times
def multiply(a, b):
    # base case is: number a is added one time
    if b == 1:
        return a
    # recursive case is: number a is added (b - 1) times
    # to this, we add one more a
    return multiply(a, b - 1) + a
```

An improvement

The improvement comes from the observation that, if the second argument is small, then we need to make fewer recursive calls. The following code enforces that.

```python
def multiply(a, b):
    if a < b:
        return multiply(b, a)
    if b == 1:
        return a
    return multiply(a, b - 1) + a
```

Problem-6

There are n companies and n candidates. Each company has selected exactly one candidate to intern at its office during the summer of 2022. We call a selection `perfect` if no two companies have selected the same candidate. An example of a perfect selection for `n = 3` is given below:

```
selection = {
'company-1': 'candidate-3',
'company-2': 'candidate-1',
'company-3': 'candidate-2'
}
```

An imperfect matching would be:

```
selection = {
'company-1': 'candidate-2',
'company-2': 'candidate-1',
'company-3': 'candidate-2'
}
```

Write a function that accepts the dictionary `selection` as input. It should return `True` if the selection is perfect and `False` otherwise.

Answer

Basic solution

```python
def is_perfect(selection):
    n = len(selection)
    the_set = set()
    for candidate in selection.values():
        the_set.add(candidate)
    uniq = len(the_set)
    return uniq == n
```

One-liner

```python
def is_perfect(selection):
    return len(set(selection.values())) == len(selection)
```

Problem-7

A non-increasing sequence of integers is said to be *shrinking* if the difference of successive terms of the sequence is strictly decreasing. For example:

$$100, 50, 40, 35, 33, 32$$

This is a *shrinking* sequence because the sequence of differences is a strictly decreasing sequence:

$$50, 10, 5, 2, 1$$

Write a function named `is_shrinking` that accepts a list (sequence) of integers as input. It should return `True` if it is a *shrinking* sequence and `False` otherwise.

Note

- Assume that the input sequence is non-empty and non-increasing.
- To get the difference, subtract the current element from the previous element.

Answer

```python
# Get difference of successive elements
# By difference, we mean previous element minus current element
# Assume non-empty, decreasing sequence
def get_diff(L):
    diff = [ ]
    # prev element in the list
    prev = L[0]
    for elem in L[1:]:
        # (prev - elem) is the difference between successive elements
        diff.append(prev - elem)
        prev = elem
    return diff

def is_shrinking(seq):
    L = get_diff(seq)
    if L == [ ]:
        # This happens if there is only one element in seq
        # In such a case, we return true as a default value
        return True
    prev = L[0]
    for elem in L[1:]:
        if elem >= prev:
            # >= is necessary because of the strictly decreasing condition
            return False
        prev = elem
    return True
```

Problem-8

Write a recursive function to find the remainder when a positive integer a is divided by a positive integer b. You can only use $+$ and $-$ operators. You are not allowed to use the $/$, $//$ or $\%$ symbols!

Answer

Basic idea is that division is repeated subtraction.

```python
def remainder(a, b):
    if a < b:
        return a
    return remainder(a - b, b)
```

Problem-9

`L` is a list that contains the scores of n students in a Mathematics test. Find the following information:

- class average
- median marks
- mode or the most frequently occurring mark; if there are multiple candidates for the mode, return the smallest among them

Use first principles. Try to avoid using built-in functions or list methods as much as possible.

Answer

Sample list of 25 marks to test the code:

```
import random
L = [ ]
for i in range(25):
    L.append(random.randint(1, 100))
```

- class average

```
def average(L):
    S = 0
    for mark in L:
        S += mark
    return S / len(L)
```

- median marks; assume that the list has an odd number of elements

```
# insert an element into a sorted list
def insert(x, L):
    out_L = [ ]
    inserted = False
    for elem in L:
        if not inserted and x < elem:
            out_L.append(x)
            inserted = True
        out_L.append(elem)
    if not inserted:
        out_L.append(x)
    return out_L
# sort the list of elements recursively
def sort(L):
    if len(L) <= 1:
        return L
    return insert(L[0], sort(L[1:]))
# find the median
def median(L):
    sorted_L = sort(L)
    # for odd number of elements
    mid = len(sorted_L) // 2
    return sorted_L[mid]
```

Note the use of functions here to break down a complex problem into simpler problems. Of course, we could have just used `sort` method to sort the list. But, we don't learn much that way, do we!

- mode

```
def mode(L):
    # use this to find the frequency of elements in L
    # key is mark, value is freq of occurrence
    P = dict()
    for mark in L:
        if mark not in P:
            P[mark] = 0
        P[mark] += 1
    # the mode is now just the key which has the greatest value
    # simple code to find the key with greatest value
    m, mode_value = -1, -1
    keys = sort(list(P.keys())) # we are using our own sort function
    for key in keys:
        value = P[key]
        if value > mode_value:
            mode_value = value
            m = key
    return m
```

Problem-10

Given a positive integer n, find the largest value of k such that the following inequality is satisfied:

$$2^k <= n$$

Input	Output
10	3
100	6
1000	9

Write two different implementations of the same function:

- iterative
- recursive

Answer

```python
def greatest_i(n):
    k = 0
    while 2 ** k <= n:
        k += 1
    return k - 1

# start with greatest_r(n, 0)
def greatest_r(n, k):
    if 2 ** k == n:
        return k
    if 2 ** k < n:
        return greatest_r(n, k + 1)
    return k - 1
```

Problem-11

`scores` is a list that has the runs scored by a batsman in all cricket matches that he has played in his career. Answer the following questions:

- How many matches has he played?
- How many centuries (hundred runs or more) has he scored?
- When was the first time that he scored a century? When was the last time that he scored a century?
- What is the longest gap between successive centuries?
- What is the longest streak of centuries in his career? A streak is a sequence of consecutive hundreds.
- What is his career average? For this problem, assume that the average is the total number of runs scored across all matches divided by the number of matches that he has played.
- Divide his career into two halves. In which part of his career did he have a better average?
- Find the number of unique scores in his career. Is there any particular score that he has achieved multiple times in his career? What is the maximum among these scores?
- Find the number of matches that fall in each of these score ranges. How will you store this information in Python?

Score range
$0 \leq s < 10$
$10 \leq s < 50$
$50 \leq s < 100$
$100 \leq s < 500$

Assume that the actual list will have several hundred elements. Also, every element of the list will be an integer in the range $[0, 500]$.

Answer

- Number of matches played

```python
len(scores)
```

- Number of centuries scored

```python
count = 0
for score in scores:
    if score >= 100:
        count += 1
```

- First time a century was scored

```python
for match, score in enumerate(scores):
    if score >= 100:
        break
first_century_at = match + 1
```

- Last time a century was scored

```
for match, score in enumerate(scores):
    if score >= 100:
        last_century_at = match + 1
```

- Longest gap between successive centuries.

```
def longest_gap(scores):
    '''Accepts scores as input; returns longest gap
    Returns -1 if batsman has scored less than two centuries'''
    centuries = [ ]
    for match, score in enumerate(scores):
        if score >= 100:
            centuries.append(match)
    if len(centuries) < 2:
        return -1
    prev = centuries[0]
    long = -1
    for match in centuries[1: ]:
        if match - prev > long:
            long = match - prev
        prev = match
    return long
```

- Longest streak

```
def longest_streak(scores):
    '''Accepts scores as input and returns longest streak'''
    centuries = [ ]
    for match, score in enumerate(scores):
        if score >= 100:
            centuries.append(match)
    if len(centuries) == 0:
        return 0
    prev = centuries[0]
    streak, max_streak = 1, 1
    for match in centuries[1:]:
        if match - prev == 1:
            streak += 1
            if streak > max_streak:
                max_streak = streak
        else:
            streak = 1
        prev = match
    return max_streak
```

- Average

```python
def average(scores, first, last):
    '''returns batsman's average in the duration [first, last]
       first: first match, last: last match
       both endpoints included; zero-indexing
       '''
    if last - first < 0:
        return -1   # invalid inputs
    S = 0
    for i in range(first, last + 1):
        S += scores[i]
    return S / (last - first + 1)
```

- Average in different parts

```python
first, last = 0, len(scores) - 1
mid = (first + last) // 2
first_half = average(scores, first, mid)
second_half = average(scores, mid + 1, last)
if first_half > second_half:
    print('first')
elif first_half == second_half:
    print('equal')
else:
    print('second')
```

- Unique scores; mode among these scores

```python
################################
P = dict()
for score in scores:
    if score not in P:
        P[score] = 0
    P[score] += 1
unique_scores = list(P.keys())
################################
# score_max_freq is nothing but the mode of scores
# mode is most frequently occuring value
score_max_freq, freq_max = -1, -1
for score, freq in P.items():
    if freq > freq_max:
        score_max_freq, freq_max = score, freq
print(score_max_freq)
```

- Score ranges

```python
P = dict()
P[(0, 10)] = 0
P[(10, 50)] = 0
P[(50, 100)] = 0
P[(100, 500)] = 0
for score in scores:
    if 0 <= score < 10:
        P[(0, 10)] += 1
    elif 10 <= score < 50:
        P[(10, 50)] += 1
```

```
    elif 50 <= score < 100:
        P[(50, 100)] += 1
    elif 100 <= score:
        P[(100, 500)] += 1
```

Problem-12

Write a function called `uniq` that accepts a list `L` as input and returns a list after removing all duplicates from it.

Test Cases

`L`	`uniq(L)`
[1, 2, 3, 1, 1, 4]	[1, 2, 3, 4]
['a', 'b', 'c', 'b', 'a', 'd']	['a', 'b', 'c', 'd']

Write three different implementations of the same function:

- Using the right collection, it is a single line of code.
- From first principles, just using lists and loops.
- A recursive solution.

Answer

- Single line implementation

```
def uniq(L):
    return list(set(L))
```

- From first principles

```
def uniq(L):
    out_L = [ ]
    for elem in L:
        if elem not in out_L:
            out_L.append(elem)
    return out_L
```

- Recursive implementation-1

```
def uniq(L):
    if len(L) <= 1:
        return L
    init = L[: -1]
    last = L[-1]
    if last in init:
        return uniq(init)
    else:
        return uniq(init) + [last]
```

- Recursive implementation-2

```python
def uniq(L):
    if len(L) <= 1:
        return L
    rest = L[1: ]
    first = L[0]
    if first in rest:
        return uniq(rest)
    else:
        return [first] + uniq(rest)
```

Problem-13 (Challenge)

a and b are two integers. The following statements are basic arithmetic facts:

- If d is a common divisor of a and b, then d is a divisor of $a - b$.
- If d is a common divisor of b and $a - b$, then d is a divisor of a.

Using these two facts, write a recursive function to compute the greatest common divisor (GCD) of a and b.

Answer

```python
def gcd(a, b):
    if a < b:
        return gcd(b, a)
    if a % b == 0:
        return b
    return gcd(a - b, b)
```

Problem-14

Consider a directed graph — G. G_r is the reverse graph of G obtained by reversing the edges in G. For example, if $u \rightarrow v$ is an edge in G, then $v \rightarrow u$ is an edge in G_r. Write a function `reverse` that accepts the adjacency matrix M of the graph G as input and returns the adjacency matrix M_r of the graph G_r.

Answer

```
def init_matrix(dim):
    A = [ ]
    for i in range(dim):
        A.append([ ])
        for j in range(dim):
            A[-1].append(0)
    return A

def reverse(mat):
    dim = len(mat)
    mat_rev = init_matrix(dim)
    for i in range(dim):
        for j in range(dim):
            if mat[i][j] == 1:
                mat_rev[j][i] == 1
    return mat_rev
```

Problem-15 (Challenge)

Write a function `binomial` that accepts two integers `n` and `k` as input and returns the coefficient of x^k in the algebraic expression $(1+x)^n$. Do not use the binomial formula anywhere. Treat this as a computational problem.

Answer

$$(1+x)^n = (1+x)^{n-1} \cdot (1+x)$$

If $(1+x)^{n-1} = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$, then using the above formula, we have:

$$\begin{aligned}(1+x)^n &= (a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1})(1+x) \\ &= a_0 + (a_0 + a_1)x + (a_1 + a_2)x^2 + \cdots + (a_{n-2} + a_{n-1})x^{n-1} + a_{n-1} x^n\end{aligned}$$

```
def expand(n):
    if n == 1:
        return [1, 1]   # (1 + x)^1 : [1, 1]
    old_coeff = expand(n - 1)   # (1 + x)^(n - 1), list of coeefs for this
    last = old_coeff[-1]
    size = len(old_coeff)
    new_coeff = old_coeff.copy()
    for i in range(1, size):
        new_coeff[i] += old_coeff[i - 1]
    new_coeff.append(last)
    return new_coeff

def binomial(n, k):
    coeff = expand(n)
    if k < 0 or k > n:
        return 'Invalid'
    return coeff[k]
```

Problem-16

Consider the following polynomial:

$$f(x) = a_0 + a_1x^1 + a_2x^2 + \cdots + a_nx^n$$

Write a recursive function named `poly` to evaluate the polynomial at any given input. The function will accept two arguments: `coeff` and `x0`. The coefficients will be a list:

```
coeff = [a0, a1, a2, ..., an]
```

`poly(coeff, x0)` should evaluate $f(x_0)$.

Answer

$$\begin{aligned} f(x) &= a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \\ &= a_0 + x(a_1 + a_2x + \cdots + a_nx^{n-1}) \\ &= a_0 + x(a_1 + x(a_2 + a_3x + \cdots + a_nx^{n-2})) \\ &= \vdots \end{aligned}$$

```
def poly(coeff, x0):
    if len(coeff) == 1:
        return coeff[0]
    return coeff[0] + x * poly(coeff[1: ], x0)
```