Exponent Last

def last\_digit(n1, n2):

value=\_\_\_\_\_\_\_

string\_value=\_\_\_\_\_\_\_

last\_digit=\_\_\_\_\_\_\_

return last\_digit

print(last\_digit(9,7))

Gapped Prime Pairs

def gap(g, m, n):

# your code

prime\_pair=[]

primes=[]

#Loop to find all prime numbers inclusive of the range

for i in range(m,n+1):

factor\_count=\_\_\_\_\_\_\_

for j in range(1,i+1):

\_\_\_\_\_\_\_

factor\_count+=1

if factor\_count==\_\_\_\_\_\_\_:

\_\_\_\_\_\_\_)

run=True

for i in range(len(primes)):

for j in primes:

if primes[i]!=j and \_\_\_\_\_\_\_:

prime\_pair.append\_\_\_\_\_\_\_

prime\_pair.append\_\_\_\_\_\_\_

run=False

break

if run==False:

break

print(prime\_pair)

gap(6,100,110)

Hashtag Creator

def generate\_hashtag(s):

#your code here

hashtag=\_\_\_\_\_\_\_

if s=="" or len(s)>140:

return False

else:

capitalizer=True

for i in range(0,len(s)):

if s[i]==' ':

capitalizer=\_\_\_\_\_\_\_

else:

if capitalizer:

hashtag+=\_\_\_\_\_\_\_

capitalizer=\_\_\_\_\_\_\_

else:

hashtag+=\_\_\_\_\_\_\_

return hashtag

print(generate\_hashtag('c i n'))

Pagination

#Pagination Helper Code

class PaginationHelper:

# The constructor takes in an array of items and a integer indicating

# how many items fit within a single page

def \_\_init\_\_(self, collection, items\_per\_page):

\_\_\_\_\_\_\_=collection

\_\_\_\_\_\_\_=items\_per\_page

# returns the number of items within the entire collection

def item\_count(self):

return \_\_\_\_\_\_\_

# returns the number of pages

def page\_count(self):

if len(self.collection) % \_\_\_\_\_\_\_

return \_\_\_\_\_\_\_

else:

return \_\_\_\_\_\_\_

# returns the number of items on the current page. page\_index is zero based

# this method should return -1 for page\_index values that are out of range

def page\_item\_count(self, page\_index):

number\_of\_pages=\_\_\_\_\_\_\_

items\_per\_page\_list=[]

for i in range(0,number\_of\_pages):

if number\_of\_pages==\_\_\_\_\_\_\_:

items\_per\_page\_list.append(PaginationHelper.item\_count())

else:

if i!= \_\_\_\_\_\_\_:

items\_per\_page\_list.append(self.items\_per\_page)

else:

items\_per\_page\_list.append\_\_\_\_\_\_\_

if page\_index> \_\_\_\_\_\_\_:

return -1

else:

return items\_per\_page\_list[page\_index]

# determines what page an item is on. Zero based indexes.

# this method should return -1 for item\_index values that are out of range

def page\_index(self, item\_index):

index\_per\_page\_list=[]

for i in range(0,PaginationHelper.page\_count()):

page\_index\_list=[]

Snail Traversal

def snail(snail\_map):

array=[]

n=\_\_\_\_\_\_\_\_\_

for i in snail\_map[0]: #First row

\_\_\_\_\_\_\_\_\_

for i in range(1,n): #3rd column

array.append\_\_\_\_\_\_\_\_\_

for i in range(1,-1,-1): #Bottom Row

array.append\_\_\_\_\_\_\_\_\_

for i in range(0,2):

array.append\_\_\_\_\_\_\_\_\_

print(array)

snail([[1,2,3],

[8,9,4],

[7,6,5]])

Pattern 1[TOUGHEST]

n = 6

# upper part of the heart

for i in range(\_\_\_ , n, 2):

# print first spaces

for j in range(1, n-i ,2):

print(\_\_\_\_\_\_\_\_\_)

# print first stars

for j in range(\_\_ , \_\_\_, 1):

print("\*", end="")

# print second spaces

for j in range(\_\_ , \_\_\_, 1):

print(" ", end="")

# print second stars

for j in range(1, i+1, 1):

print("\*", end="")

print()

# lower part

for i in range(n,0,-1):

for j in range(i, n, 1):

print(" ", end="")

for j in range(\_\_\_\_\_\_\_ , 1):

print(\_\_\_\_ , end="")

print()

Pattern 2

n = 5

# downward pyramid

for i in range(n-1):

for j in range(\_\_\_\_\_\_\_): #Spaces

print(' ', end='')

for k in range(\_\_\_\_\_\_\_): #Stars

print('\*', end='')

print()

# upward pyramid

for i in range(n):

for j in range(\_\_\_\_\_\_\_): #Spaces

print(' ', end='')

for k in range(\_\_\_\_\_\_\_): #Stars

print('\*', end='')

print()

Pattern 3

n = 5

# upward hollow pyramid

for i in range(n):

for j in range(\_\_\_\_\_\_\_): #Spaces

print(' ', end='')

for j in range(\_\_\_\_\_\_\_):

if j == 0 or j == \_\_\_\_\_\_\_: #Stars or spaces

print('\*', end='')

else:

print(\_\_\_\_\_\_\_ , end='')

print()

# downward pyramid

for i in range(\_\_\_\_\_\_\_):

for j in range(\_\_\_\_\_\_\_): #Spaces

print(' ', end='')

for j in range(\_\_\_\_\_\_\_): #Stars or Spaces

if j == 0 or j == 2\*(n - i - 1) - 2:

print('\*', end='')

else:

print(' ', end='')

print()