Codificación y Programación.

Quiz Capítulos #5, #6 Y #7.

Q. 05-01. ¿Cuántas comparaciones se ejecutaron en el siguiente proceso de clasificación por inserción?

```
def bubblesort(S):
    n = len(S)
    for i in range(n):
        print(S)
        for j in range(n - 1):
            if S[j] > S[j + 1]:
                S[j], S[j + 1] = S[j + 1], S[j]
```

```
S = [50, 30, 40, 10, 20]
bubblesort(S)
print(S)
```

```
[50, 30, 40, 10, 20]
[30, 40, 10, 20, 50]
[30, 10, 20, 40, 50]
[10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]
```

Realizó 20 comparaciones

Q. 05-02. ¿Cuántas comparaciones se ejecutaron en el siguiente proceso de clasificación por inserción?

```
def insertionsort2(S):
    n = len(S)
    for i in range(1, n):
        print(S)
        x = S[i]
        j = i - 1
        while j >= 0 and S[j] > x:
            S[j + 1] = S[j]
            j -= 1
        S[j + 1] = x
```

```
S = [50, 30, 40, 10, 20]
insertionsort2(S)
print(S)
```

```
[50, 30, 40, 10, 20]
[30, 50, 40, 10, 20]
[30, 40, 50, 10, 20]
[10, 30, 40, 50, 20]
[10, 20, 30, 40, 50]
```

Realizó 8 comparaciones

Q. 05-03. ¿Cuántas veces se ejecutó la función merge2() en el siguiente proceso de clasificación por fusión?

```
S = [6, 2, 11, 7, 5, 4, 8, 16, 10, 3]
mergesort2(S, 0, len(S) - 1)
print(S)
```

Realizó 20 comparaciones

Q. 05-04. Dada la lista a continuación, escriba la salida después de ejecutar la función partición1().

```
S = [15, 10, 12, 20, 25, 13, 22]
partition1(S, 0, len(S) - 1)
print(S)
```

```
[15, 10, 12, 20, 25, 13, 22] 0 6 pivot =  15
[13, 10, 12, 15, 25, 20, 22]
```

Realizó 6 comparaciones

Q. 06-01. Diseñe un algoritmo que halle la función factorial de cualquier número empleando recursividad

```
def factorial(num):
    if num < 0:
        print("No existe, debe sert positivo")

    elif num == 0:
        return 1

    else:
        fact = 1
        while(num > 1):
            fact *= num
            num -= 1
        return fact

num = int(input("Digite un número: "));

print("Número factorial de",num,"es", factorial(num))
```

```
Digite un número: 5
Factorial of 5 is 120
```

Q. 06-02. Diseñe un algoritmo que halle la función factorial de cualquier número empleando memoización

```
def factorial(n):
    return 1 if (n==1 or n==0) else n * factorial(n - 1);

num = int(input("Digite un número: "));
print("Número factorial de",num,"es", factorial(num))
```

```
Digite un número: 5
Factorial of 5 is 120
```

Q. 06-03. Diseñe un algoritmo que codifique la secuencia de Fibonacci empleando recursividad

```
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1) + fib(n-2)

fib(8)
```

```
21
```

Q. 06-04. Ejercicio lógico: en cada palabra, reemplace las letras por un número, teniendo en cuenta que para cada palabra separada por un espacio la suma de sus dígitos es un número al cuadrado Encuentra el número representado por cada letra.

1]:
```python
from math import sqrt, floor
import time

def is_square_digitsum(n):
    s = 0
    while n > 0:
        s += n % 10
        n //= 10
    if sqrt(s) == int(sqrt(s)):
        return True
    return False

def find_all_squares():
    sqrs = [[] for _ in range(5)]
    for i in range(1, floor(sqrt(10 ** 5)) + 1):
        n = i * i
        if not is_square_digitsum(n):
            continue
        s = str(n)
        if len(s) == 3 and s[1] != s[2]:
            continue
        if len(s) == 5 and s[2] != s[3]:
            continue
        if len(s) in  [4,5] and len(set(s)) != 4:
            continue
        sqrs[len(s) - 1].append(n)
    return sqrs

def promising(s, n, dic):
    for i in range(len(s)):
        digit = int(str(n)[i])
        for key, value in dic.items():
            if key == s[i] and value != digit:
                return False
            if value == digit and key != s[i]:
                return False
        return True
```

```
def solve(words, dic, squares):
    global solved
    if (len(words) == 0):
        solved = dic
    else:
        s = words[0]
        candidates = squares[len(s) - 1]
        for n in candidates:
            if promising(s, n, dic):
                newdic = dic.copy()
                for i in range(len(s)):
                    newdic[s[i]] = int(str(n)[i])
                solve(words[1:], newdic, squares)

def main():
    squares = find_all_squares()
    print(squares)
    words = ["A", "MERRY", "XMAS", "TO", "ALL"]
    dic = {}
    solve(words, dic, squares)
    for word in words:
        print(word, end=": ")
        for c in word:
            print(solved[c], end="")
        print()

start = time.time()
solved = {}
main()
end = time.time()
print("Elapsed Time: ", end - start, " seconds")
```

```
[[1, 4, 9], [36, 81], [100, 144, 400, 900], [2304, 2601, 3481, 7396, 9025], [27556, 34225, 52441, 61009, 62001, 70225]]
A: 9
MERRY: 32001
XMAS: 7396
TO: 81
ALL: 900
Elapsed Time:  0.000997304916381836  seconds
```

A MERRY XMAS TO ALL

A	M	E	R	R	Y	X	M	A	S	T	O	A	L	L

- **Código de referencia**

```
from math import sqrt, floor
import time

def is_square_digitsum(n):
    s = 0
    while n > 0:
        s += n % 10
        n //= 10
    if sqrt(s) == int(sqrt(s)):
        return True
    return False

def find_all_squares():
    sqrs = [[] for _ in range(5)]
    for i in range(1, floor(sqrt(10 ** 5)) + 1):
        n = i * i
        if not is_square_digitsum(n):
            continue
        s = str(n)
```

- **Código de referencia**

```
        if len(s) == 3 and s[1] != s[2]:
            continue
        if len(s) == 5 and s[2] != s[3]:
            continue
        if len(s) in [4, 5] and len(set(s)) != 4:
            continue
        sqrs[len(s) - 1].append(n)
    return sqrs

def promising(s, n, dic):
    for i in range(len(s)):
        digit = int(str(n)[i])
        for key, value in dic.items():
            if key == s[i] and value != digit:
                return False
            if value == digit and key != s[i]:
                return False
    return True

def solve(words, dic, squares):
    global solved
    if (len(words) == 0):
        solved = dic
    else:
        s = words[0]
        candidates = squares[len(s) - 1]
        for n in candidates:
            if promising(s, n, dic):
                newdic = dic.copy()
                for i in range(len(s)):
                    newdic[s[i]] = int(str(n)[i])
                solve(words[1:], newdic, squares)

def main():
    squares = find_all_squares()
    # print(squares)
```

- **Código de referencia**

```
        if len(s) == 3 and s[1] != s[2]:
            continue
        if len(s) == 5 and s[2] != s[3]:
            continue
        if len(s) in [4, 5] and len(set(s)) != 4:
            continue
        sqrs[len(s) - 1].append(n)
    return sqrs

def promising(s, n, dic):
    for i in range(len(s)):
        digit = int(str(n)[i])
        for key, value in dic.items():
            if key == s[i] and value != digit:
                return False
            if value == digit and key != s[i]:
                return False
    return True

def solve(words, dic, squares):
    global solved
    if (len(words) == 0):
        solved = dic
    else:
        s = words[0]
        candidates = squares[len(s) - 1]
        for n in candidates:
            if promising(s, n, dic):
                newdic = dic.copy()
                for i in range(len(s)):
                    newdic[s[i]] = int(str(n)[i])
                solve(words[1:], newdic, squares)

def main():
    squares = find_all_squares()
    # print(squares)
```

- **Código de referencia**

```
    words = ['A', 'TO', 'ALL', 'XMAS', 'MERRY']
    dic = {}
    solve(words, dic, squares)
    for word in words:
        print(word, end=": ")
        for c in word:
            print(solved[c], end="")
        print()

start = time.time()
solved = {}
main()
end = time.time()
print("Elapsed Time: ", end - start, " seconds")
```

Q. 07-01. Convierta los siguientes datos del diccionario en un objeto dataframe usando Pandas. (El objeto del marco de datos se denomina df.)

```
d = {'col1': [1, 2], 'col2': [3, 4], 'col3': [5, 6], 'col4': [7, 8]}
```

```
In [19]: import pandas as pd
         d = {'col1':[1,2],'col2':[3,4],'col3':[5,6],'col4':[7,8]}

         df = pd.DataFrame(d)
         df
```

Out[19]:

	col1	col2	col3	col4
0	1	3	5	7
1	2	4	6	8

Q. 07-02. Para el dataframe creado en la Pregunta 1, cree un nuevo dataframe que consista sólo en los datos de la columna con el nombre de columna 'col4'. (El dataframe se denomina new_df.)

```
In [25]: new_df = pd.DataFrame(d['col4'],columns=['col4'])
         new_df
```

Out[25]:

	col4
0	7
1	8

Q. 07-03. El índice del dataframe creado en la Pregunta 1 es 0,1. Escriba un comando para cambiar el nombre de estos índices a primero y segundo.

```
In [27]: df = pd.DataFrame(d,index=['primero','segundo'])
         df
```

Out[27]:

	col1	col2	col3	col4
primero	1	3	5	7
segundo	2	4	6	8

Q. 07-04. Escriba un comando para buscar datos faltantes en el dataframe df creado en la Pregunta 1 e imprima el resultado. (Sin embargo, los datos que faltan deben devolverse como verdaderos).

```
In [28]: df.isnull()
```

Out[28]:

	col1	col2	col3	col4
primero	False	False	False	False
segundo	False	False	False	False

Q. 07-05. Escriba un comando para verificar el resumen de las estadísticas descriptivas (desviación estándar, valor mínimo, moda, etc.) del marco de datos df creado en la pregunta 1 e imprima el resultado.

```
In [31]: df.std(axis = 0, skipna = True)
```

```
Out[31]: col1    0.707107
         col2    0.707107
         col3    0.707107
         col4    0.707107
         dtype: float64
```

```
In [34]: stats = df.describe()
         stats
```

Out[34]:

	col1	col2	col3	col4
count	2.000000	2.000000	2.000000	2.000000
mean	1.500000	3.500000	5.500000	7.500000
std	0.707107	0.707107	0.707107	0.707107
min	1.000000	3.000000	5.000000	7.000000
25%	1.250000	3.250000	5.250000	7.250000
50%	1.500000	3.500000	5.500000	7.500000
75%	1.750000	3.750000	5.750000	7.750000
max	2.000000	4.000000	6.000000	8.000000