Módulo 2 - Capítulo 4

Unidad 22

Quiz 1

Escriba la salida de los códigos del siguiente ejemplo que usa una pila.

```
stack=Stack()
stack.push('Banana')
stack.push('Apple')
stack.push('Tomato')
stack.pop()
stack.push('Strawberry')
stack.push('Grapes')
stack.pop()
print(stack.stack)
```

```
class Stack():
    def __init__(self):
        self.stack = []

    def is_empty(self):
        return len(self.stack) == 0

    def push(self, item):
        self.stack.append(item)

    def pop(self):
        return None if self.is_empty() else self.stack.pop()

    def peek(self):
        return None if self.is_empty() else self.stack[-1]
```

```
stack = Stack()
print('stack = Stack()          -->', stack.stack, end="\n\n")

stack.push('Banana')
print("stack.push('Banana')     -->", stack.stack)

stack.push('Apple')
print("stack.push('Apple')      -->", stack.stack)

stack.push('Tomato')
print("stack.push('Tomato')     -->", stack.stack)
```

```
stack.pop()
print("stack.pop()              -->", stack.stack)

stack.push('Strawberry')
print("stack.push('Strawberry') -->", stack.stack)

stack.push('Grapes')
print("stack.push('Grapes')     -->", stack.stack)

stack.pop()
print("stack.pop()              -->", stack.stack)
```

```
stack = Stack()          --> []

stack.push('Banana')     --> ['Banana']
stack.push('Apple')      --> ['Banana', 'Apple']
stack.push('Tomato')     --> ['Banana', 'Apple', 'Tomato']
stack.pop()              --> ['Banana', 'Apple']
stack.push('Strawberry') --> ['Banana', 'Apple', 'Strawberry']
stack.push('Grapes')     --> ['Banana', 'Apple', 'Strawberry', 'Grapes']
stack.pop()              --> ['Banana', 'Apple', 'Strawberry']
```

Quiz 2

Escriba la salida de los códigos del siguiente ejemplo que usa una pila.

```
stack = Stack()

items = [10*i for i in range(1, 10)]

for item in items:
    stack.push(items)
    if (item // 10) % 2 == 0:
        stack.pop()

print(stack.stack)
```

```
stack = Stack()
print('stack = Stack()', stack.stack, end="\n\n")

items = [10*i for i in range(1, 10)]

for item in items:
    stack.push(item)
    print('stack.push(items) -->', f'item: {item} -->', stack.stack)
```

```
    if (item // 10) % 2 == 0:
        stack.pop()
        print('      stack.pop() -->', f'item: {item} -->', stack.stack)

print(f'\n{stack.stack}')
```

```
stack = Stack() []

stack.push(items) --> item: 10 --> [10]
stack.push(items) --> item: 20 --> [10, 20]
      stack.pop() --> item: 20 --> [10]
stack.push(items) --> item: 30 --> [10, 30]
stack.push(items) --> item: 40 --> [10, 30, 40]
      stack.pop() --> item: 40 --> [10, 30]
stack.push(items) --> item: 50 --> [10, 30, 50]
stack.push(items) --> item: 60 --> [10, 30, 50, 60]
      stack.pop() --> item: 60 --> [10, 30, 50]
stack.push(items) --> item: 70 --> [10, 30, 50, 70]
stack.push(items) --> item: 80 --> [10, 30, 50, 70, 80]
      stack.pop() --> item: 80 --> [10, 30, 50, 70]
stack.push(items) --> item: 90 --> [10, 30, 50, 70, 90]

[10, 30, 50, 70, 90]
```

Quiz 3

El documento HTML consta de muchas etiquetas, como se muestra a continuación. Esciba un programa que coincida con las etiquetas de documentos HTML.

```
<html>
    <body>
        <h1>Hello, World!</h1>

        <p> We are learning the art of coding with Python programming language.
Here we are learning ... </p>

        <ul>
            <li> Data Structures, </li>
            <li> Algorithms, </li>
            <li> and Computational Thinking, eventually. </li>
        </ul>
    </body>
</html>
```

TIP:

En Python, el método `find()` se usa para ubicar la posición inicial de la subcadena. Encuentre `<` y `>` para distingir las etiquteas de documentos HTML.

Si la etiqueta distinguida incluye `/`, entonces es una etiqueta cerrada, y si no, es una etiqueta abierta. Verifique la coincidencia de la etiqueta HTML de una manera similar al uso de una pila para hacer coincidir los paréntesis.

Cadena de prueba 1: `<html><body> <h1> hello world </h1>`

Cadena de prueba 2: `<html><body><h1>Hello, World!</h1>`
`<p> We are learning the art of coding with Python programming language.`
`Here we are learning ... </p><ul><li> Data Structures, </li><li> Algorithms, </li>`
`<li> and Computational Thinking, eventually. </li></ul></body></html>`

```
text = input('Ingrese la cadena del documento HTML: ')
```

```
start = text.find('<')

while start != -1:
    end = text.find('>', start + 1)
    tag = text[start : end + 1]
    print(tag, end=' ')
    start = text.find('<', end + 1)
```

```
<html> <body> <h1> </h1>
```

```
opened_tags = [
    '<html>',
    '<body>',
    '<h1>',
    '<p>',
    '<ul>',
    '<li>'
]

closed_tags = [
    '</html>',
    '</body>',
    '</h1>',
    '</p>',
    '</ul>',
    '</li>'
]

tag_pairs = {
```

```
    closed: opened for opened, closed in zip(opened_tags, closed_tags)
}
```

```
def extract_tags(text: str):
    start = text.find('<')
    tags = []

    while start != -1:
        end = text.find('>', start + 1)
        tag = text[start: end + 1]
        tags.append(tag)
        start = text.find('<', end + 1)

    return tags
```

```
def validate_tags():
    stack = Stack()
    is_valid = True

    text = input('Ingrese la cadena del documento HTML')
    tags = extract_tags(text)
    print('Tags reconocidos:', tags)

    for tag in tags:
        if tag in opened_tags:
            stack.push(tag)
        elif tag in closed_tags:
            if stack.is_empty() or stack.pop() != tag_pairs[tag]:
                is_valid = False
                break

    if not stack.is_empty():
        is_valid = False

    print("El código ingresado es {}".format('valido' if is_valid else
'invalido'))
```

```
validate_tags()
```

```
Tags reconocidos: ['<html>', '<body>', '<h1>', '</h1>', '<p>', '</p>', '<ul>',
'<li>', '</li>', '<li>', '</li>', '<li>', '</li>', '</ul>', '</body>', '</html>']
El código ingresado es valido
```

```
validate_tags()
```

```
Tags reconocidos: ['<html>', '<body>', '<h1>', '</h1>']
El código ingresado es invalido
```

Unidad 23

Quiz 1

Escriba la salida de los códigos del ejemplo que usa una cola.

```
queue=Queue()
queue.enqueue('Banana')
queue.enqueue('Apple')
queue.enqueue('Tomato')
queue.dequeue()
queue.enqueue('Strawberry')
queue.enqueue('Grapes')
queue.dequeue()
print(queue.queue)
```

```
class Queue():
    def __init__(self):
        self.queue = []

    def is_empty(self):
        return len(self.queue) == 0

    def enqueue(self, item):
        self.queue.append(item)

    def dequeue(self):
        return None if self.is_empty() else self.queue.pop(0)

    def size(self):
        return len(self.queue)
```

```
queue = Queue()
print("queue = Queue()             -->", queue.queue)

queue.enqueue('Banana')
print("queue.enqueue('Banana')     -->", queue.queue)
```

```
queue.enqueue('Apple')
print("queue.enqueue('Apple')      -->", queue.queue)

queue.enqueue('Tomato')
print("queue.enqueue('Tomato')     -->", queue.queue)

queue.dequeue()
print("queue.dequeue()            -->", queue.queue)

queue.enqueue('Strawberry')
print("queue.enqueue('Strawberry') -->", queue.queue)

queue.enqueue('Grapes')
print("queue.enqueue('Grapes')     -->", queue.queue)

queue.dequeue()
print("queue.dequeue()            -->", queue.queue)
```

```
queue = Queue()             --> []
queue.enqueue('Banana')     --> ['Banana']
queue.enqueue('Apple')      --> ['Banana', 'Apple']
queue.enqueue('Tomato')     --> ['Banana', 'Apple', 'Tomato']
queue.dequeue()             --> ['Apple', 'Tomato']
queue.enqueue('Strawberry') --> ['Apple', 'Tomato', 'Strawberry']
queue.enqueue('Grapes')     --> ['Apple', 'Tomato', 'Strawberry', 'Grapes']
queue.dequeue()             --> ['Tomato', 'Strawberry', 'Grapes']
```

Quiz 2

Escriba la salida de los códigos del ejemplo que usan una cola.

```
queue=Queue()
items=[10*i for i in range(1,11)]
for item in items:
  queue.enqueue(items)
  if (item // 10) % 2 == 0:
    queue.dequeue()
print(queue.queue)
```

```
queue = Queue()
print('queue = Queue()', queue.queue, end="\n\n")

items = [10*i for i in range(1, 11)]

for item in items:
    queue.enqueue(item)
```

```
    print('queue.enqueue(items) -->', f'item: {item} -->', queue.queue)

    if (item // 10) % 2 == 0:
        queue.dequeue()
        print('     queue.dequeue() -->', f'item: {item} -->', queue.queue)


print(f'\n{queue.queue}')
```

```
queue = Queue() []

queue.enqueue(items) --> item: 10 --> [10]
queue.enqueue(items) --> item: 20 --> [10, 20]
     queue.dequeue() --> item: 20 --> [20]
queue.enqueue(items) --> item: 30 --> [20, 30]
queue.enqueue(items) --> item: 40 --> [20, 30, 40]
     queue.dequeue() --> item: 40 --> [30, 40]
queue.enqueue(items) --> item: 50 --> [30, 40, 50]
queue.enqueue(items) --> item: 60 --> [30, 40, 50, 60]
     queue.dequeue() --> item: 60 --> [40, 50, 60]
queue.enqueue(items) --> item: 70 --> [40, 50, 60, 70]
queue.enqueue(items) --> item: 80 --> [40, 50, 60, 70, 80]
     queue.dequeue() --> item: 80 --> [50, 60, 70, 80]
queue.enqueue(items) --> item: 90 --> [50, 60, 70, 80, 90]
queue.enqueue(items) --> item: 100 --> [50, 60, 70, 80, 90, 100]
     queue.dequeue() --> item: 100 --> [60, 70, 80, 90, 100]

[60, 70, 80, 90, 100]
```

Quiz 3

Consulte la definición de clase de la siguiente manera para completar y probar la clase `Deque`

```
class Deque:
    def __init__(self):
        self.queue = []

    def is_empty(self):
        return len(self.queue) == 0

    def add_first(self, item):
        self.queue.insert(0, item)

    def remove_first(self):
        return None if self.is_empty() else self.queue.pop(0)

    def add_last(self, item):
```

```
        self.queue.append(item)

    def remove_last(self):
        return None if self.is_empty() else self.queue.pop()
```

Una cola es un tipo de dato abstracto, donde solo es posible agreagar en un lado y solo es posible eliminar en el otro lado.

Por el contrario, la cola de doble terminación (`Deque`) es la estructura de datos en la que es posoble agregar y eliminar en ambos lados.

```
deque = Deque()
print("deque = Queue()              -->", deque.queue)

deque.add_first('Banana')
print("deque.add_first('Banana')    -->", deque.queue)

deque.add_first('Apple')
print("deque.add_first('Apple')     -->", deque.queue)

deque.add_first('Tomato')
print("deque.add_first('Tomato')    -->", deque.queue)

deque.remove_first()
print("deque.remove_first()         -->", deque.queue)

deque.add_last('Strawberry')
print("deque.add_last('Strawberry') -->", deque.queue)

deque.add_last('Grapes')
print("deque.add_last('Grapes')     -->", deque.queue)

deque.remove_last()
print("deque.remove_last()          -->", deque.queue)
```

```
deque = Queue()              --> []
deque.add_first('Banana')    --> ['Banana']
deque.add_first('Apple')     --> ['Apple', 'Banana']
deque.add_first('Tomato')    --> ['Tomato', 'Apple', 'Banana']
deque.remove_first()         --> ['Apple', 'Banana']
deque.add_last('Strawberry') --> ['Apple', 'Banana', 'Strawberry']
deque.add_last('Grapes')     --> ['Apple', 'Banana', 'Strawberry', 'Grapes']
deque.remove_last()          --> ['Apple', 'Banana', 'Strawberry']
```

Unidad 24

Quiz 1

Adivina la salida del siguiente algoritmo. Analice el código y describa el comportamiento de este algoritmo.

```
def find_two(nums):
    x = y = 0

    for i in range(1, len(nums)):
        if nums[x] < nums[i]:
            x = i
        elif nums[y] > nums[i]:
            y = i

    return x, y
```

```
nums = [11, 27, 45, 26, 59, 28, 17, 53]

i, j = find_two(nums)

print(nums[i], nums[j])
```

```
59 11
```

El algoritmo retorna el índice del número más grande y el índice el número más pequeño. Inicia asignado a la variable `x` el valor de `y`, quien a su vez tiene el valor de `0`. Luego, recorre el arreglo de números en el rango de `1` hasta el tamaño del arreglo (excluyente). Lo siguiente es comparar si el número en la posición `x` es menor al número de la posición que se está iterando, en cuyo caso se reemplaza `x` por la variable de iteración, pero si no se cumple esta condición, realiza otra evaluación para saber si el número en la posición de `y` es mayor al número de la posición que se está iterando, en cuyo caso se reemplaza `y` por la variable de iteración.

De esta manera, `x` obtendrá el máximo valor, y `y` será el valor mínimo.

Quiz 2

¿Cuántas comparaciones se deben hacer para `find_two()` que se implementó en el quiz 1?

```
def find_two(nums):
    x = y = 0

    for i in range(1, len(nums)):
        if nums[x] < nums[i]:
            print('comparación')
            x = i
        elif nums[y] > nums[i]:
            print('comparación')
```

```
            y = i

    return x, y
```

```
nums = [11, 27, 45, 26, 59, 28, 17, 53]

i, j = find_two(nums)

print(nums[i], nums[j])
# print(f'Cantidad de comparaciones: {counter}')
```

```
comparación
comparación
comparación
59 11
```

Se realizan `len(nums) - 1` comparaciones, ya que este el número de iteraciones en el intervalo de `[1, len(nums))`, el cual se usa en la función `range` del ciclo `for`

Quiz 3

Escriba un programa para contar cuántas veces se usó una palabra específica en la oración ingresada por el usuario.

En la oración ingresada por el usuario, una palabra se distingue por la presencia de espacios.

- Use la función `input().split()` para crear la lista `S` que tiene cada cadena como su elemento.
- Use la función `input()` para recibir la palabra que el usuario buscará y guárdela en `x`.
- La función `word_count()` recibe `S` y `x` como parámetros y proporciona un retorno contando cuántas `x` están presentes en la `S`.

```
def word_count(s: list[str], x: str) -> int:
    counter = sum([1 for i in range(len(s)) if s[i].lower() == x.lower()])
    return counter
```

```
s = list(input('Input a sentence: ').split())
x = input('Input a word to search: ')

count = word_count(s, x)

print(f'In the sentence "{" ".join(s)}", the word "{x}" is appeared {count}
times.')
```

```
In the sentence "The quick brown fox jumps over the lazy dog", the word "the" is
appeared 2 times.
```

Unidad 25

Quiz 1

El siguiente es el código para el juego de "adivinar números". Si `maximum = 100` y `number = 51`, ¿Cuántos conteos se imprimirían?

```
def game():
    maximum = int(input('Ingrese el número máximo: '))
    number = int(input('Ingrese su número a adivinar: '))
    count = 0

    low,high = 1,maximum

    while low < high:
        mid = (low + high) // 2
        count += 1
        print(f'low: {low}, mid: {mid}, high: {high}')

        if mid == number:
            print(f'\nSu número es {number}')
            break
        elif mid > number:
            high = mid - 1
        else:
            low = mid + 1

    print(f'Se tiene un total de {count} veces')
```

```
game()
```

```
low: 1, mid: 50, high: 100
low: 51, mid: 75, high: 100
low: 51, mid: 62, high: 74
low: 51, mid: 56, high: 61
low: 51, mid: 53, high: 55
low: 51, mid: 51, high: 52

Su número es 51
Se tiene un total de 6 veces
```

Quiz 2

Si `maximum = 100` y `number = 25` en el juego anterior, ¿Cuántos conteos se imprimirían?

```
game()
```

```
low: 1, mid: 50, high: 100
low: 1, mid: 25, high: 49

Su número es 25
Se tiene un total de 2 veces
```

Quiz 3

Si se proporciona una lista ordenada de número y un número entero aleatorio `x`, implemente la función que devuelve la ubicación del índice donde se insertará `x`. Sin embargo, `S` debe ser una lista ordenada incluso después de insertar `x`.

```
def search_insert_position(nums: list[int], x: int):
    pos = 0
    temp_list = nums.copy()
    temp_list.append(x)
    temp_list.sort()

    low, high = 0, len(temp_list) - 1

    while low < high:
        mid = (low + high) // 2

        if temp_list[mid] == x:
            pos = mid
            break
        elif temp_list[mid] > x:
            high = mid - 1
        else:
            low = mid + 1
    return pos
```

```
nums = [10, 20, 40, 50, 60, 80]
x = int(input('Ingrese un número para ser insertado: '))

pos = search_insert_position(nums, x)

print(f'x={x} debe ser insertado en la posición {pos}.')
```

```
nums.insert(pos, x)
print(nums)
```

```
x=70 debe ser insertado en la posición 5.
[10, 20, 40, 50, 60, 70, 80]
```

Unidad 26

Quiz 1

Utilice la función `hash` de la tabla hash para calcular la clave y la clave hash de `Alice in Wonderland`

```
class HashTable:

    def __init__(self, size):
        self.size = size
        self.table = {}

        for i in range(size):
            self.table[i] = []

    def hash(self, key):
        return key % self.size

    def get(self, key):
        return self.table[self.hash(key)]

    def put(self, key, value):
        bucket = self.table[self.hash(key)]
        if value not in bucket:
            bucket.append(value)
```

```
table = HashTable(8)

book = 'Alice in Wonderland'

key = sum(map(ord, book))

table.put(key, book)

print(f'key={key}, value={table.get(key)}, bucket={table.hash(key)}')
```

```
key=1795, value=['Alice in Wonderland'], bucket=3
```

Quiz 2

Si hay 10 ranuras en la estantería, con el siguiente código encuentre las ranuras donde se colocarán cada uno de los siguiente libros:

```
table = HashTable(10)

books = [
    'The Little Prince',
    'The Old Man and the Sea',
    'The Little Mermaid',
    'Beauty and the Beast',
    'The Last Leaf',
    'Alice in Wonderland'
]

for book in books:
    key = sum(map(ord, book))
    table.put(key, book)
    print(f'key={key}, hash={table.hash(key)}, bucket={table.get(key)}')

print()

for key in table.table.keys():
    print(key, table.table[key])
```

```
key=1584, hash=4, bucket=['The Little Prince']
key=1929, hash=9, bucket=['The Old Man and the Sea']
key=1678, hash=8, bucket=['The Little Mermaid']
key=1837, hash=7, bucket=['Beauty and the Beast']
key=1133, hash=3, bucket=['The Last Leaf']
key=1795, hash=5, bucket=['Alice in Wonderland']

0 []
1 []
2 []
3 ['The Last Leaf']
4 ['The Little Prince']
5 ['Alice in Wonderland']
6 []
7 ['Beauty and the Beast']
8 ['The Little Mermaid']
9 ['The Old Man and the Sea']
```

Quiz 3

Cree un convertidor que convierta números arábigos en números romanos utilizando la tabla hash que se proporciona a continuación:

```
table = {
    1000: 'M', 900: 'CM',
    500: 'D', 400: 'CD',
    100: 'C', 90: 'XC',
    50: 'L', 40: 'XL',
    10: 'X', 9: 'IX',
    5: 'V', 4: 'IV',
    1: 'I'
}
```

```
def int_to_roman(number: int)-> str:
    result = ''

    for value_key, letter in table.items():
        while number >= value_key:
            result += letter
            number -= value_key

    return result
```

```
num = int(input('Ingrese un número: '))
print(f'num = {num} -> {int_to_roman(num)}')
```

```
num = 9753 -> MMMMMMMMMDCCLIII
```

Módulo 2 - Capítulo 5

Unidad 27

Quiz 1

¿Cuántas operaciones de intercambio se han ejecutado en el proceso de ordenamiento de burbujas siguiente?

```
def bubble(S):
    n = len(S)
    exchange_counter = 0

    for _ in range(n):
        print(S)
        for j in range(n - 1):
            if S[j] > S[j + 1]:
                exchange_counter += 1
                print('\t>>> exchange_counter:', exchange_counter)
                S[j], S[j + 1] = S[j + 1], S[j]

    print('\n>>> Final exchange_counter:', exchange_counter)

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

```
[50, 30, 40, 10, 20]
    >>> exchange_counter: 1
    >>> exchange_counter: 2
    >>> exchange_counter: 3
    >>> exchange_counter: 4
[30, 40, 10, 20, 50]
    >>> exchange_counter: 5
    >>> exchange_counter: 6
[30, 10, 20, 40, 50]
    >>> exchange_counter: 7
    >>> exchange_counter: 8
[10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]

>>> Final exchange_counter: 8
```

Quiz 2

¿Cuántas operaciones de comparación se han ejecuado en el proceso de ordenamiento por inserción que se muestra a continuación?

```python
def insertion_sort(S):
    n = len(S)
    comparation_counter = 0

    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
            comparation_counter += 1
            print('\t>>> comparation_counter:', comparation_counter)

        S[j + 1] = x

    print('\n>>> Final comparation_counter:', comparation_counter)


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

```
[50, 30, 40, 10, 20]
    >>> comparation_counter: 1
[30, 50, 40, 10, 20]
    >>> comparation_counter: 2
[30, 40, 50, 10, 20]
    >>> comparation_counter: 3
    >>> comparation_counter: 4
    >>> comparation_counter: 5
[10, 30, 40, 50, 20]
    >>> comparation_counter: 6
    >>> comparation_counter: 7
    >>> comparation_counter: 8

>>> Final comparation_counter: 8
```

Quiz 3

Dadas dos palabras, escribe un algoritmo que determine si estas dos cadenas son anagramas. Un anagrama es una palabra que se forma reordenando las letras de otra palabra utilizando todas las letras originales exactamente una vez. (Por ejemplo, `LISTEN` y `SILENT` son anagramas).

```python
def convert_to_ascci(S: str):
    return [ ord(c) for c in S ]
```

```
def insertion_sort(S):
    n = len(S)

    for i in range(1, n):
        x = S[i]
        j = i - 1

        while j >= 0 and S[j] > x:
            S[j + 1] = S[j]
            j -= 1
        S[j + 1] = x

    return S

def is_anagram(a, b, sort_fun = insertion_sort):
    a = sort_fun(convert_to_ascci(a.lower()))
    b = sort_fun(convert_to_ascci(b.lower()))

    print("¿Las cadenas son anagramas? =>", a == b)


a = "lIsTeN"
b = "SiLeNt"

is_anagram(a, b)
```

```
¿Las cadenas son anagramas? => True
```

```
a = input("Ingrese la cadena a: ")
b = input("Ingrese la cadena b: ")

is_anagram(a, b)
```

```
¿Las cadenas son anagramas? => True
```

Quiz 4

El uso de un algoritmo de clasificación permite determinar fácilmente si se trata de un anagrama o no.

- Cree una función que evalúe los anagramas utilizando la función `sorted()` incorporada de python
- Modificar la función `selection_sort()` para crear una función que determine los anagramas
- Modificar la función `insertion_sort()` para crear una función que determine los anagramas

```

```

```
def selection_sort(S):
    n = len(S)
    for i in range(n - 1):
        smallest = i
        for j in range(i + 1, n):
            if S[j] < S[smallest]:
                smallest = j
        S[i], S[smallest] = S[smallest], S[i]
    return S
```

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

```
a = "lIsTeN"
b = "SiLeNt"

is_anagram(a, b, sort_fun=sorted)
is_anagram(a, b, sort_fun=selection_sort)
is_anagram(a, b, sort_fun=insertion_sort)

a = "lIsTeN"
b = "SiLeNcE"

is_anagram(a, b, sort_fun=sorted)
is_anagram(a, b, sort_fun=selection_sort)
is_anagram(a, b, sort_fun=insertion_sort)
```

```
¿Las cadenas son anagramas? => True
¿Las cadenas son anagramas? => True
¿Las cadenas son anagramas? => True
¿Las cadenas son anagramas? => False
¿Las cadenas son anagramas? => False
¿Las cadenas son anagramas? => False
```

Unidad 28

Quiz 1

¿Cuántas veces se ejecutó la función `merge_sort()` en el proceso de ordenamiento por mezcla que se muestra a continuación?

```
def merge(S, low, mid, high):
    R = []
    i, j = low, mid + 1

    for _ in range(low, high+1):
        if i > mid:
            R.append(S[j])
            j += 1
        elif j > high:
            R.append(S[i])
            i += 1
        elif S[i] < S[j]:
            R.append(S[i])
            i += 1
        else:
            R.append(S[j])
            j += 1
    for k in range(len(R)):
        S[low] = R[k]
        low += 1

execution_counter = 0

def merge_sort(S, low, high):
    global execution_counter
    execution_counter += 1
    print('\t>>> S:', S)
    print('\t>>> execution_counter:', execution_counter)

    if low < high:
        mid = (low + high) // 2
        merge_sort(S, low, mid)
        merge_sort(S, mid + 1, high)
        merge(S, low, mid, high)

def evaluate_execution(S):
    global execution_counter
    execution_counter = 0

    merge_sort(S, 0, len(S) - 1)
    print('>>> S:', S)
    print('>>> Final execution_counter:', execution_counter)
```

```
S = [6, 2, 11, 7, 5, 4, 8, 16, 10, 3]
evaluate_execution(S)
```

```
    >>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3]
    >>> execution_counter: 1
    >>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3]
    >>> execution_counter: 2
    >>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3]
    >>> execution_counter: 3
    >>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3]
    >>> execution_counter: 4
    >>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3]
    >>> execution_counter: 5
    >>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3]
    >>> execution_counter: 6
    >>> S: [2, 6, 11, 7, 5, 4, 8, 16, 10, 3]
    >>> execution_counter: 7
    >>> S: [2, 6, 11, 7, 5, 4, 8, 16, 10, 3]
    >>> execution_counter: 8
    >>> S: [2, 6, 11, 7, 5, 4, 8, 16, 10, 3]
    >>> execution_counter: 9
    >>> S: [2, 6, 11, 7, 5, 4, 8, 16, 10, 3]
    >>> execution_counter: 10
    >>> S: [2, 5, 6, 7, 11, 4, 8, 16, 10, 3]
    >>> execution_counter: 11
    >>> S: [2, 5, 6, 7, 11, 4, 8, 16, 10, 3]
    >>> execution_counter: 12
    >>> S: [2, 5, 6, 7, 11, 4, 8, 16, 10, 3]
    >>> execution_counter: 13
    >>> S: [2, 5, 6, 7, 11, 4, 8, 16, 10, 3]
    >>> execution_counter: 14
    >>> S: [2, 5, 6, 7, 11, 4, 8, 16, 10, 3]
    >>> execution_counter: 15
    >>> S: [2, 5, 6, 7, 11, 4, 8, 16, 10, 3]
    >>> execution_counter: 16
    >>> S: [2, 5, 6, 7, 11, 4, 8, 16, 10, 3]
    >>> execution_counter: 17
    >>> S: [2, 5, 6, 7, 11, 4, 8, 16, 10, 3]
    >>> execution_counter: 18
    >>> S: [2, 5, 6, 7, 11, 4, 8, 16, 10, 3]
    >>> execution_counter: 19
>>> S: [2, 3, 4, 5, 6, 7, 8, 10, 11, 16]
>>> Final execution_counter: 19
```

```
S = [6, 2, 11, 7, 5, 4, 8, 16, 10, 3, 1, 12, 9]
evaluate_execution(S)
```

```
>>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 1
>>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 2
>>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 3
>>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 4
>>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 5
>>> S: [6, 2, 11, 7, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 6
>>> S: [2, 6, 11, 7, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 7
>>> S: [2, 6, 11, 7, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 8
>>> S: [2, 6, 11, 7, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 9
>>> S: [2, 6, 7, 11, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 10
>>> S: [2, 6, 7, 11, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 11
>>> S: [2, 6, 7, 11, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 12
>>> S: [2, 6, 7, 11, 5, 4, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 13
>>> S: [2, 6, 7, 11, 4, 5, 8, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 14
>>> S: [2, 4, 5, 6, 7, 8, 11, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 15
>>> S: [2, 4, 5, 6, 7, 8, 11, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 16
>>> S: [2, 4, 5, 6, 7, 8, 11, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 17
>>> S: [2, 4, 5, 6, 7, 8, 11, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 18
>>> S: [2, 4, 5, 6, 7, 8, 11, 16, 10, 3, 1, 12, 9]
>>> execution_counter: 19
>>> S: [2, 4, 5, 6, 7, 8, 11, 10, 16, 3, 1, 12, 9]
>>> execution_counter: 20
>>> S: [2, 4, 5, 6, 7, 8, 11, 3, 10, 16, 1, 12, 9]
>>> execution_counter: 21
>>> S: [2, 4, 5, 6, 7, 8, 11, 3, 10, 16, 1, 12, 9]
>>> execution_counter: 22
>>> S: [2, 4, 5, 6, 7, 8, 11, 3, 10, 16, 1, 12, 9]
>>> execution_counter: 23
>>> S: [2, 4, 5, 6, 7, 8, 11, 3, 10, 16, 1, 12, 9]
>>> execution_counter: 24
>>> S: [2, 4, 5, 6, 7, 8, 11, 3, 10, 16, 1, 12, 9]
```

```
    >>> execution_counter: 25
>>> S: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16]
>>> Final execution_counter: 25
```

Quiz 2

Dadas N listas ordenadas como entrada, escribe un programa que las fusione en una lista ordenada

```
def multiple_merge(numbers_list):
    if len(numbers_list) == 1:
        return numbers_list[0]
    else:
        mid = len(numbers_list) // 2
        left = multiple_merge(numbers_list[:mid])
        right = multiple_merge(numbers_list[mid:])
        return merge(left, right)


def merge(left, right):
    merged_list = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged_list.append(left[i])
            i += 1
        else:
            merged_list.append(right[j])
            j += 1
    merged_list += left[i:]
    merged_list += right[j:]
    return merged_list
```

```
N = int(input("Ingrese el número de listas: "))
numbers_list = []

for i in range(N):
    numbers = list(map(int, input("Ingrese una lista de números: ").split()))
    print(numbers)

    numbers_list.append(numbers)

ordered_list = multiple_merge(numbers_list)
print("Fusionada dentro: ", ordered_list)
```

```
[9, 8, 7]
[6, 5, 4]
```

```
[3, 2, 1]
Fusionada dentro:  [3, 2, 1, 6, 5, 4, 9, 8, 7]
```

Unidad 29

Quiz 1

Dada la siguiente lista, escriba la salida después de ejecutar la función `partition()`

```
def partition(S, low, high):
    pivot = S[low]
    left, right = low + 1, high

    while left < right:
        while left <= right and S[left] <= pivot:
            left += 1
        while left <= right and S[right] >= pivot:
            right -= 1

        if left < right:
            S[left], S[right] = S[right], S[left]

    pivot_point = right
    S[low], S[pivot_point] = S[pivot_point], S[low]

    return pivot_point

def quick_sort(S, low, high):
    if low < high:
        pivot_point = partition(S, low, high)
        print('\t>>> pivot_point:', pivot_point)
        quick_sort(S, low, pivot_point - 1)
        quick_sort(S, pivot_point + 1, high)
```

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

```
    >>> pivot_point: 3
    >>> pivot_point: 2
    >>> pivot_point: 1
    >>> pivot_point: 6
    >>> pivot_point: 5
[10, 12, 13, 15, 20, 22, 25]
```

Quiz 2

Escriba un algoritmo que encuentre el K° elemento mayor, dados N elementos desordenados.

Después de resolver el problema con los dos métodos anteriores, analiza qué algoritmo es más eficiente.

- Puede devolver el elemento K° después de utilizar la función de ordenamiento.
- Puede utilizar la función partition() para realizar una llamada recursiva hasta que el pivote sea el K° elemento.

```
from random import randint
import time

random_numbers = 0
k = 3

def measure_time(func):
    global random_numbers
    random_numbers = [randint(0, 10_000) for _ in range(10_000)]
    def measure_function(*args, **kwargs):
        start = time.time()
        nums = func(*args, **kwargs)
        k_number = nums[-k]
        end = time.time()
        print('>>> Execution time:', end - start)
        print('>>> k_number:', k_number)
        return k_number
    return measure_function
```

Buble Sort

```
@measure_time
def bubble(S):
    n = len(S)

    for _ in range(n):
        for j in range(n - 1):
            if S[j] > S[j + 1]:
                S[j], S[j + 1] = S[j + 1], S[j]

    return S

bubble(random_numbers)
```

```
>>> Execution time: 8.513029098510742
>>> k_number: 9998
9998
```

Insertion Sort

```
@measure_time
def insertion_sort(S):
    n = len(S)

    for i in range(1, n):
        x = S[i]
        j = i - 1

        while j >= 0 and S[j] > x:
            S[j + 1] = S[j]
            j -= 1

        S[j + 1] = x

    return S

insertion_sort(random_numbers)
```

```
>>> Execution time: 2.000645160675049
>>> k_number: 10000
10000
```

Selection Sort

```
@measure_time
def selection_sort(S):
    n = len(S)
    for i in range(n - 1):
        smallest = i
        for j in range(i + 1, n):
            if S[j] < S[smallest]:
                smallest = j
        S[i], S[smallest] = S[smallest], S[i]
    return S

selection_sort(random_numbers)
```

```
>>> Execution time: 1.975259780883789
>>> k_number: 9997
9997
```

Merge Sort

```
def merge(S, low, mid, high):
    R = []
    i, j = low, mid + 1

    for _ in range(low, high+1):
        if i > mid:
            R.append(S[j]); j += 1
        elif j > high:
            R.append(S[i]); i += 1
        elif S[i] < S[j]:
            R.append(S[i]); i += 1
        else:
            R.append(S[j]); j += 1
    for k in range(len(R)):
        S[low] = R[k]; low += 1

def merge_sort(S, low, high):
    if low < high:
        mid = (low + high) // 2
        merge_sort(S, low, mid)
        merge_sort(S, mid + 1, high)
        merge(S, low, mid, high)
    return S

@measure_time
def exe_merge_sort():
    return merge_sort(random_numbers, 0, len(random_numbers) - 1)

exe_merge_sort()
```

```
>>> Execution time: 0.028768539428710938
>>> k_number: 9998
9998
```

Quick Sort

```
def partition(S, low, high):
    pivot = S[low]
    left, right = low + 1, high

    while left < right:
        while left <= right and S[left] <= pivot:
            left += 1
        while left <= right and S[right] >= pivot:
            right -= 1
```

```
        if left < right:
            S[left], S[right] = S[right], S[left]

    pivot_point = right
    S[low], S[pivot_point] = S[pivot_point], S[low]

    return pivot_point


def quick_sort(S, low, high):
    if low < high:
        pivot_point = partition(S, low, high)
        quick_sort(S, low, pivot_point - 1)
        quick_sort(S, pivot_point + 1, high)

    return S


@measure_time
def exe_quick_sort():
    return quick_sort(random_numbers, 0, len(random_numbers) - 1)


exe_quick_sort()
```

```
>>> Execution time: 0.013513326644897461
>>> k_number: 9999
9999
```

Despues de comparar los tiempos de ejecución en los diversos métodos de ordenamiento, he determinado que el mejor método para un arreglo con 10.000 de números aleatorios es el método `quick_sort()` con un tiempo de ejecución equivalente a `0.0135` segundos.

Módulo 2 - Capítulo 6

Unidad 30

Quiz 1

En el problema del intercambio de monedas, supongamos que hay una moneda de 400 wones.

Si es así, escriba el resultado de cómo el algoritmo `coin_change()` determinará el cambio de la moneda de 800 wones.

```
def coin_change(coins, amount):
    changes = []
    largest = 0

    while(amount > 0):
        if amount < coins[largest]:
            largest += 1
        else:
            changes.append(coins[largest])
            amount -= coins[largest]

    return changes
```

```
coins = [500, 400, 100, 50, 10]
amount = int(input('Ingrese la cantidad:'))

changes = coin_change(coins, amount)
print('Cantidad:', amount)
print(changes, len(changes))
```

```
Cantidad: 800
[500, 100, 100, 100] 4
```

Tomando como base el algoritmo redactado en la Unidad 30, si se tiene una moneda con una denominación de 400 wones, el código retornará 4 monedas: 1 de 500 y 3 de 100.

Quiz 2

Suponga que el número de monedas es en problema no es infinito. Por ejemplo, si tiene las siguiente monedas en su billetera, ¿Cómo debe distribuir 710 won como cambio?

Billetera:

```
{
    500: 1,
    100: 1,
    50: 3,
    10: 2
}
```

Cambio:

```
{
    500: 1,
    100: 1,
    50: 2,
    10: 1
}
```

Lo primero es comenzar con la moneda de mayor valor, en este caso 500 won. Se debe comparar si la cantidad a cambiar es menor que el valor de la moneda, en cuyo caso se pasa a la moneda de la siguiente denominación. Si la cantidad a cambiar es mayor o igual a la moneda actual, se resta la moneda de la billetera y se aumenta la cantidad de monedas en el cambio. Por último se debe repetir el proceso anterior hasta que el cambio sea de 0 o ya no haya monedas en la billetera.

Como resultado en la billetera solo deben quedar las siguientes monedas:

```
{
    500: 0,
    100: 0,
    50: 1,
    10: 1
}
```

Quiz 3

Dada la cantidad de monedas y la cantidad de cambio, encuentre la cantidad mínima de monedas que puede devolver como cambio.

```
def coin_change(change, coins):
    result = {}
    coins_keys = sorted(coins.keys(), reverse=True)

    for coin in coins_keys:
        coins_counter = min(change // coin, coins[coin])
        change -= coins_counter * coin
        coins[coin] -= coins_counter
```

```
            if coins_counter:
                result[coin] = coins_counter

        return result


coins = {500: 1, 100: 1, 50: 3, 10: 2}
change_amount = 710

print(coin_change(change_amount, coins))
```

```
{500: 1, 100: 1, 50: 2, 10: 1}
```