Status	Finished
Started	Tuesday, 19 November 2024, 10:11 AM
Completed	Tuesday, 19 November 2024, 10:13 AM
Duration	2 mins 28 secs
Marks	9.00/9.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Strings

Las strings representan secuencias contiguas de caracteres. Los valores literales de tipo string (*str*), se encierran entre comillas dobles o simples:

```
var1 = "hola"
var2 = 'hola'
```

```
str_var = "Hello world"
print(str_var)          # Muestra "Hello world" (sin las comillas delimitadoras)
```

Se puede crear una string multilínea usando triples comillas (simples o dobles):

```
var3 = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
```

(los caracteres de salto de línea serán parte de la string).

La longitud de una string (número de caracteres) se conoce usando la función `len()`:

```
str_var = "Hello world"
print(len(str_var))     # Muestra el valor entero 11
```

Information

Secuencias de escape

Se pueden introducir caracteres especiales en las strings (tabuladores, saltos de línea, caracteres por código, ...) usando secuencias de escape, que son secuencias de caracteres que empiezan con el carácter '\' (llamado barra invertida o *backslash*).

\' - Comilla simple (evita confusión con comilla simple delimitadora)

```
print('Hello\' World!')
```
```
Hello' World!
```

\" - Comilla doble (evita confusión con comilla doble delimitadora)

```
print("Hello\" World!")
```
```
Hello" World!
```

\\ - backslash (evita confusión con el propio backslash de secuencia de escape)

```
print("Hello\\ World!")
```
```
Hello\ World!
```

\a - Sonido de campana

```
print("Hello\a World!")
```
```
Hello World!
```

\b - Backspace (retroceso)

```
print("Hello\b World!")
```
```
Hell World!
```

\v - Vertical tab

```
print("Hello\v World!")
```
```
Hello
      World!
```

\xhh - Carácter por valor hexadecimal

```
print("Hello\x40 World!")
```
```
Hello@ World!
```

\n - Newline (nueva línea)

```
print("Hello\n World!")
```
```
Hello
 World!
```

\N{name} - Carácter Unicode por nombre

```
print("Hello\N{BABY ANGEL} World!")
```
```
Hello👼 World!
```

\ooo - Carácter por valor octal

```
print("Hello\100 World!")
```
```
Hello@ World!
```

\r - Carriage return (al mostrar la string, sobrescribe desde el principio de la línea actual)

```
print("Hello\r World!")
```
```
 World!
```

\uxxxx - Carácter Unicode por valor hexadecimal (16 bits)

```
print("Hello\u041b World!")
```
```
HelloЛ World!
```

\Uxxxxxxxx - Carácter Unicode por valor hexadecimal (32 bits)

```
print("Hello\U000001a9 World!")
```
```
HelloƩ World!
```

Information

Mayúsculas y minúsculas

Existen métodos para saber si las letras que contiene una string son mayúsculas o minúsculas y para hacer cambios de un caso al otro.

El método **`isupper()`** devuelve True si las letras de una string son mayúsculas; el método **`islower()`** hace lo propio si son minúsculas.

```
test_string = "en un lugar de La Mancha de cuyo nombre"
a = test_string.isupper()    # False
```

El método **`upper()`** devuelve una nueva string, como la original, pero con sus letras transformadas a mayúsculas; lo contrario hacen los métodos **`lower()`** y **`casefold()`** (ambos transforman a minúsculas).

```
b = test_string.upper()      # "EN UN LUGAR DE LA MANCHA DE CUYO NOMBRE"
```

El método **`capitalize()`** devuelve una nueva string, como la original, pero poniendo en mayúscula la letra inicial y en minúscula el resto.

```
c = test_string.capitalize() # "En un lugar de la mancha de cuyo nombre"
```

El método **`title()`** devuelve una nueva string igual que la original pero poniendo en mayúscula la letra inicial de cada palabra y en minúscula el resto. Existe un método llamado **`istitle()`** para verificar si una string sigue este formato.

```
d = test_string.title()      # "En Un Lugar De La Mancha De Cuyo Nombre"
```

El método **`swapcase()`** devuelve una nueva string invirtiendo mayúsculas y minúsculas.

```
f = d.swapcase()             # "eN uN lUGAR dE lA mANCHA dE cUYO nOMBRE"
```

Question 1

Complete

Mark 1.00 out of 1.00

¿Qué valor tiene la variable *d*?

```
test_string = "tres tristes tigres comían trigo en un trigal"
a = test_string.upper()
b = a.title()
c = b.swapcase()
d = c.capitalize()
```

Select one:

- TRES TRISTES TIGRES COMÍAN TRIGO EN UN TRIGAL
- Tres Tristes Tigres Comían Trigo En Un Trigal
- "Tres tristes tigres comían trigo en un trigal"
- "tRES tRISTES tIGRES cOMÍAN tRIGO eN uN tRIGAL"

Information

Categorías de caracteres

Existen una serie de métodos para conocer qué tipo de caracteres forman una string.

isalnum()

Devuelve True si todos los caracteres de la string son alfanuméricos.

isalpha()

Devuelve True si todos los caracteres de la string están en el alfabeto.

isdecimal()

Devuelve True si todos los caracteres de la string son decimales.

isdigit()

Devuelve True si todos los caracteres de la string son dígitos.

isidentifier()

Devuelve True si la string es un identificador. Una string se considera un identificador válido si sólo contiene letras alfanuméricas (a-z, A-Z y 0-9), o guiones bajos (_). Un identificador válido no puede comenzar con un número o contener espacios.

isnumeric()

Devuelve True si todos los caracteres de la string son numéricos

isprintable()

Devuelve True si todos los caracteres de la string son imprimibles.

isspace()

Devuelve True si todos los caracteres de la string son espaciadores (espacios, tabuladores, saltos de línea, ...)

Question 2

Complete

Mark 1.00 out of 1.00

¿Cuál de las siguientes instrucciones muestra *True*?

Select one:

- print("agua y aceite".isdecimal())
- print("agua y aceite".isalpha())
- print("agua y aceite".isprintable())
- print("agua y aceite".isalnum())

Information

Acceso a los componentes de una string

Se puede acceder a un carácter concreto de una string escribiendo la string, o el nombre de la variable que la referencia, seguida del índice de la posición que ocupa el carácter deseado entre corchetes. El índice de la primera posición es 0.

```
str_var = "Hello world"
print(str_var[0])       # Muestra (sin comillas) el primer carácter de str_var:'H'
print(str_var[4])       # Muestra (sin comillas) el quinto carácter de str_var:'o'
```

También se pueden usar índices relativos a la longitud de la string:

```
str_var = "Hello world"
print(str_var[-1])      # Muestra el último carácter de str_var:'d'
print(str_var[-5])      # Muestra el quinto carácter, empezando por el final, de str_var:'w'
```

Se puede obtener una substring (subsecuencia) de una string usando el operador de segmento, o slice, ([:]). El segmento deseado se indica mediante el índice del primer elemento y el índice de la posición siguiente al último elemento. Si se omite el segundo valor, se toma el segmento desde el primer índice hasta el final; si se omite el primero, toma desde el principio hasta la posición del índice anterior al indicado:

```
str_var = "Hello world"
print(str_var[2:5])     # Muestra los caracteres de 3º al 5º: "llo"
print(str_var[2:])      # Muestra los caracteres a partir del 3º: "llo world"
print(str_var[:5])      # Muestra los caracteres hasta la 5ª posición: "Hello"
```

Usando un paso de -1 se puede obtener la string invertida:

```
str_var = "Hello world"
print(str_var[::-1])    # Muestra "dlrow olleH"
```

Question 3

Complete

Mark 1.00 out of 1.00

¿Qué valor se asigna a la variable substring?

```
               01234567890123456789012345678901234
test_string = "En un lugar de la mancha de cuyo..."
substring = test_string[8:16]
```

Answer: gar de l

Information

Concatenación de strings

El diccionario de la RAE define concatenar como "unir dos o más cosas"; en el caso de las strings, concatenar dos strings consiste en formar una nueva string juntando, en orden, dos preexistentes.

En Python existen dos operadores de concatenación: el signo más (**+**) es el operador de concatenación de strings y el asterisco (*****) es el operador de repetición (equivalente a realizar una concatenación repetidas veces).

```
str_var = "Hello world"
print('a' + 'a')        # Muestra "aa"
print(str_var + "TEST") # Muestra el valor de str_var concatenado con "TEST": "Hello worldTEST"
print('ab' * 5)         # Muestra "abababababab"
print(str_var * 2)      # Muestra "Hello worldHello world"
```

Podríamos decir que la repetición de strings es a la concatenación de strings lo que la multiplicación de números enteros es a la suma de números enteros.

Nótese que la concatenación junta directamente las strings, sin añadir ningún carácter en medio.

Question 4

Complete

Mark 1.00 out of 1.00

¿Qué muestra el siguiente programa?

```
              01234567890123456789012345678901234
test_string = "En un lugar de La Mancha de cuyo..."
substring = test_string[0:2] + " " + test_string[15:24]
print(substring)
```

Answer: En La Mancha

Information

Localización de una substring

El método **`find()`** permite localizar una substring dentro de una string:

```
s1 = "Esto es una string de prueba"
print(s1.find("una"))  # 8
```

El método `find()` devuelve el índice del primer carácter de la primera aparición de la string pasada como parámetro ("una" en el ejemplo) en la string sobre la que se aplica (la referenciada por `s1` en el ejemplo).

```
s1 = "Esto es una string, es una secuencia, pero no una lista"
p = s1.find("una") # 8
```

En caso de que la string a buscar no se encuentre en la string donde se busca, el método `find()` devuelve el valor `-1`.

También se puede usar el método **`index()`**, aplicable a cualquier tipo de secuencia (strings, listas, tuplas, ...).

```
s1 = "Esto es una string de prueba"
print(s1.index("una"))  # 8
```

La diferencia es que, si la string buscada no se encuentra, el método `index()` produce un error. Una forma de evitarlo es usar el operador `in` para comprobar primero que está la string buscada:

```
s1 = "Esto es una string de prueba"
if "una" in s1
    print(s1.index("una"))
```

Si sólo nos interesa saber si la string buscada está o no en la string donde se busca, pero no necesitamos conocer su índice, basta con el operador `in`:

```
print("una" in s1)  # True
```

El método **`find()`** admite dos parámetros adicionales opcionales:

- `start`: índice de la posición donde comenzar la búsqueda (el valor por omisión es 0)
- end: índice de la posición donde dejar de buscar (el valor por omisión es la longitud de la string donde se busca)

Por ejemplo:

```
print(s1.find("e ", 10, 24))  # 20
```

Otros métodos de localización

Los métodos **`rfind()`** y **`rindex()`** localizan la última ocurrencia de la string buscada, a diferencia de sus contrarios, `find()` e `index()`, que localizan la primera.

El método **`startswith()`** devuelve `True` si la string pasada como parámetro coincide con el comienzo de la string sobre la que se busca. Lo equivalente hace **`endswith()`** por el final.

El método **`count()`** devuelve el número de apariciones de la string buscada en la string de búsqueda.

El método **`partition()`** devuelve una tupla con las tres partes en que podemos considerar dividida la string en que busca: la substring anterior, la primera ocurrencia de la substring encontrada y la substring posterior; si no se encuentra, las tres partes serán la propia string buscada y dos strings vacías. El método **`rpartition()`** hace lo mismo desde el final, con la última ocurrencia de la string buscada.

Question 5

Complete

Mark 1.00 out of 1.00

¿Qué valor muestra el siguiente código?

```
          01234567890123456789012345678901234567890123456
str_var = "El policía alto dio el alto al ladrón en el alto"
print (str_var[15:].find("alto"))
```

Answer: 8

Information

El método `join()`

El método `join()` devuelve una string concatenando las sucesivas strings contenidas en la secuencia pasada como parámetro, usando como separador la string con la que se llama al método.

Ejemplo:

```
words = ['sun', 'is', 'shining']
espacio = ' ' # un espacio
message = espacio.join(words)
```

```
print(message)                  # "sun is shining" (sin comillas)

import random
numbers = [str(random.randint(1, 9)) for i in range(5)]
print(numbers)                  # p.e. ['9', '1', '9', '3','2']
excercise = ' + '.join(numbers)
print(excercise, '= ')          # "9 + 1 + 9 + 3 + 2 = " (sin comillas)
```

En los ejemplos anteriores, las secuencias pasadas al método `join()` son listas, pero pueden ser cualquier clase de secuencia, por ejemplo, otra string, en cuyo caso, el separador se intercalaría entre los caracteres de la misma:

```
s1 = "ABC"
print("-".join(s1)) # "A-B-C" (sin comillas)
```

Information

El método `split()`

El método `split()` crea una lista cuyos elementos son sucesivos trozos de la string sobre la que se aplica.

Si se le llama sin parámetros, los elemento de la lista son los trozos que en la string original están **delimitados** por espaciadores (*whitespaces*), esto es, cualesquiera separadores estándar de texto, tales como espacios en blanco o/y tabuladores:

```
a = 'esto es un ejemplo'
b = a.split()  # ['esto', 'es', 'un', 'ejemplo']
```

Se puede especificar un **separador** entre los trozos (se consideraría que existe un trozo string vacía entre dos separadores seguidos). En el siguiente ejemplo es la coma:

```
a = 'esto es, un ejemplo, de split, con parámetro'
b = a.split(',')  # ['esto es', ' un ejemplo', ' de split', ' con parámetro']
```

También se puede especificar, con un parámetro adicional, el número máximo de "cortes" a realizar. El siguiente ejemplo devuelve una lista con tres elementos, dado que se pasa un 2 como segundo parámetro:

```
a = 'esto es, un ejemplo, de split, con parámetros'
b = a.split(',', 2)  # ['esto es', ' un ejemplo', ' de split, con parámetros']
```

Question 6

Complete

Mark 1.00 out of 1.00

El usuario teclea una frase en la consola y el programa le responde mostrando un mensaje:

```
output = ' '.join(input('enter your sentence: ').split()[::-1])
print(output)
```

¿Cuál es el mensaje mostrado para la siguiente entrada?

The Earth is my home.

Select one:

- The Earth is my home.
- The-1Earth-1is-1my-1home.
- home. my is Earth The
- La primera línea de código es incorrecta, se produce un error

Question 7

Complete

Mark 1.00 out of 1.00

Dada la siguiente asignación, empareje cada uso de *split()* con su resultado.

```
str_var = "El policía alto dio el alto al ladrón en el alto"
```

str_var.split()	['El', 'policía', 'alto', 'dio', 'el', 'alto', 'al', 'ladrón', 'en', 'el', 'alto']
str_var.split('i')	['El pol', 'cía alto d', 'o el alto al ladrón en el alto']
str_var.split('alto')	['El policía ', ' dio el ', ' al ladrón en el ', '']

Information

El método `replace()`

El método `replace()`, devuelve una string que es igual a la string sobre la que se aplica, cambiando las ocurrencias de una substring por otro valor.

Supongamos la siguiente asignación:

```
a = "Susanita tiene un ratón"
```

Supongamos ahora que el ratón de Susanita se escapó:

```
b = a.replace("tiene", "tenía") # b = "Susanita tenía un ratón"
```

El método `replace()` sustituye cada una de las apariciones:

```
a = "Un globo, dos globos, tres globos"
b = a.replace("globo", "elefante") # b = "un elefante, dos elefantes, tres elefantes"
```

Question 8

Complete

Mark 1.00 out of 1.00

¿Qué valor se asigna a la variable *b*?

```
a = "camisas"
b = a.replace("sas", "rones").replace("cami", "perche")
```

Answer: percherones

Information

Recorte y relleno: los métodos `strip()`, `lstrip()`, `rstrip()`, `ljust()`, `rjust()`, `center()` y `zfill()`

A veces tenemos una string con espacios al principio o al final que no nos resultan útiles y queremos eliminar. El método **`strip()`** elimina los espaciadores de los extremos de una string:

```
a = "    string con espacios   "
b = a.strip() # "string con espacios"
```

Los métodos **`lstrip()`** y **`rstrip()`** hacen lo mismo, pero sólo en uno de los extremos (izquierdo - `left` o derecho - `right`):

```
left = a.lstrip()   # "string con espacios   "
right = a.rstrip()  # "    string con espacios"
```

También existen métodos para justificar una string en un tamaño determinado, ajustándola a la izquierda o a la derecha y rellenando el espacio sobrante con un carácter especificado:

```
a = "hola"
b = a.ljust(10, "_") # "hola______"
c = a.rjust(10, "_") # "______hola"
```

Combinando los métodos `ljust()` y `rjust()`, se puede lograr una justificación "centrada", aunque existe un método, `center()`, para hacer eso:

```
mid = 5 + len(a) // 2
d = a.ljust(mid, "_").rjust(10, "_") # "___hola___"
d1 = a.center(10, "_")               # "___hola___"
```

Por su parte, el método **`zfill()`** rellena una string añadiéndole ceros por la izquierda hasta alcanzar un tamaño determinado:

```
num = "12"
num1 = num.zfill(5) # "00012"
```

Obviamente, siempre es posible rellenar con cualquier carácter usando operaciones más básicas:

```
num = "12"
num1 = 'X' * (5 - len(num)) + num # "XXX12"
```

Question 9

Complete

Mark 1.00 out of 1.00

¿Qué valor se asigna a *num1*?

```
       0123456789
num = "    12    "
num1 = num.lstrip().zfill(12)
```

Select one:

- `"00    12    "`
- `"000000000012"`
- `"00000012    "`