1. Create Regular Expressions that:

a. Recognize following strings bit, but, bat, hit, hat or hut

```
import re
words = "hello bit but bat hit hat hut and not different words hit"
pattern = r'(b[iau]t|h[iau]t)'
matches = re.findall(pattern,words)

print(matches)
```

Output:

```
['bit', 'but', 'bat', 'hit', 'hat', 'hut', 'hit']
```

Explanations:

The given pattern is a regular expression (regex) designed to match certain words with specific characters. Let's break down the regex pattern r'(b[iau]t|h[iau]t)':

1. r'': This denotes a raw string in Python, where backslashes are treated literally, and not as escape characters.
2. (): Parentheses are used for grouping.
3. b[iau]t: This part matches any string that starts with 'b', followed by either 'i', 'a', or 'u', and then 't'.
    - b: Matches the character 'b'.
    - [iau]: Matches any one of the characters 'i', 'a', or 'u'.
    - t: Matches the character 't'.
4. |: This is a logical OR operator in regex, meaning it will match either the expression on its left or the expression on its right.
5. h[iau]t: This part matches any string that starts with 'h', followed by either 'i', 'a', or 'u', and then 't'.
    - h: Matches the character 'h'.
    - [iau]: Matches any one of the characters 'i', 'a', or 'u'.
    - t: Matches the character 't'.

b. Match any pair of words separated by a single space, that is, first and last names.

```
import re
text = """
Vishal Sharma
Sapna Sharma
Shlok Sharma
```

```
"""
name_pattern = r'\b(\w+)\s(\w+)\b'
names = re.findall(name_pattern, text)
print(names)
```

Output:

```
[('Vishal', 'Sharma'), ('Sapna', 'Sharma'), ('Shlok', 'Sharma')]
```

Explanations:

The given pattern is a regular expression (regex) designed to match names in a specific format, typically first name followed by last name. Let's break down the regex pattern r'\b(\w+)\s(\w+)\b':

1. r'': This denotes a raw string in Python, where backslashes are treated literally, and not as escape characters.
2. \b: This asserts a word boundary, ensuring the match is at the beginning of a word.
3. (\w+): This captures a sequence of one or more word characters (letters, digits, and underscores).
    - \w: Matches any word character (equivalent to [a-zA-Z0-9_]).
    - +: Matches one or more of the preceding token.
    - () (parentheses): Capture the matched sequence as a group.
4. \s: Matches any whitespace character (spaces, tabs, etc.).

c. Match any word and single letter separated by a comma and single space, as in last name, first initial.

```
import re
text = """
Sharma, V
Sharma, S
Sharma, S
"""
name_pattern = r'\b(\w+),\s(\w)\b'
names = re.findall(name_pattern, text)
print(names)
```

Output:

```
[('Sharma', 'V'), ('Sharma', 'S'), ('Sharma', 'S')]
```

d. Match simple Web domain names that begin with www. and end with a ".com" suffix; for example, www.yahoo.com. Extra Credit: If your regex also supports other high-level domain names, such as .edu, .net, etc. (for example, www.foothill.edu).

```
import re
urls = [
"www.example.com",
"www.yahoo.com",
"www.foothill.edu",
"www.example.net",
"www.example.org"
]
domain_pattern = r'www\.[A-Za-z0-9.-]+\.(?:com|edu|net|org)'
matches = [url for url in urls if re.fullmatch(domain_pattern, url)]
print(matches)
```

Output:

['www.example.com', 'www.yahoo.com', 'www.foothill.edu', 'www.example.net', 'www.example.org']

The given pattern is a regular expression (regex) designed to match domain names with specific top-level domains (TLDs). Let's break down the regex pattern r'www\.[A-Za-z0-9.-]+\.(?:com|edu|net|org)':

1. r'': This denotes a raw string in Python, where backslashes are treated literally, and not as escape characters.
2. www\.: This matches the literal string "www.".
    - www: Matches the literal characters "www".
    - \.: Escapes the dot character, matching a literal dot.
3. [A-Za-z0-9.-]+: This matches one or more of any letter (uppercase or lowercase), digit, dot, or hyphen.
    - [A-Za-z0-9.-]: Matches any character in the set (letters, digits, dots, hyphens).
    - +: Matches one or more of the preceding token.
4. \.: Matches a literal dot.
5. (?:com|edu|net|org): This is a non-capturing group that matches one of the specified TLDs.
    - (?: ...) : Non-capturing group.
    - com|edu|net|org: Matches any of the TLDs "com", "edu", "net", or "org".

e. Match a street address according to your local format (keep your regex general enough to match any number of street words, including the type designation). For example, American street addresses use the format: 1180 Bordeaux Drive. Make your regex flexible enough to support multi-word street names such as: 3120 De la Cruz Boulevard.

```
import re
addresses = '''
1 raj twin bunglows near dharoi colony
12 Harikrishna residency
12 Niknath society,
7 raj nagar visnagar"
'''
address_pattern = r'\b\d+\s(?:\w+\s)*\w+\s\w+\b'
#address_pattern=r'\b\d+\s(\w+\s)*'
matches=re.findall(address_pattern,addresses)
print(matches)
```

Output :

['1 raj twin bunglows near dharoi colony', '12 Harikrishna residency', '12

Niknath society', '7 raj nagar visnagar']

The given pattern is a regular expression (regex) designed to match addresses with a specific format. Let's break down the regex pattern r'\b\d+\s(?:\w+\s)*\w+\s\w+\b':

1. r'': This denotes a raw string in Python, where backslashes are treated literally, and not as escape characters.
2. \b: This asserts a word boundary, ensuring the match is at the beginning of a word.
3. \d+: This matches one or more digits.
    - \d: Matches any digit (equivalent to [0-9]).
    - +: Matches one or more of the preceding token.
4. \s: Matches any whitespace character (spaces, tabs, etc.).
5. (?:\w+\s)*: This is a non-capturing group that matches zero or more occurrences of a word followed by a whitespace.
    - (?: ...): Non-capturing group.
    - \w+: Matches one or more word characters (letters, digits, and underscores).
    - \s: Matches any whitespace character.
    - *: Matches zero or more of the preceding token.

2. Create Regular Expressions That:

a. Extract the complete timestamps from each line.

```
import re
text = """
Timestamp: 2023-08-15 12:30:45 - Event 1
Timestamp: 2023-09-20 14:25:10 - Event 2
Timestamp: 2023-10-05 10:15:30 - Event 3
"""
# Define a regular expression pattern to extract timestamps
timestamp_pattern = r'\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}'

# Extract timestamps from each line
timestamps = re.findall(timestamp_pattern, text)

print(timestamps)
for timestamp in timestamps:
print(timestamp)
```

Output:

```
['2023-08-15 12:30:45', '2023-09-20 14:25:10', '2023-10-05 10:15:30']
2023-08-15 12:30:45
2023-09-20 14:25:10
2023-10-05 10:15:30
```

Explanations:

Your regular expression pattern r'\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}' is designed to match timestamps in the format YYYY-MM-DD HH:MM:SS. Here is a breakdown of the pattern:

- \d{4}: Matches exactly four digits (the year part).
- -: Matches a hyphen (the separator between year, month, and day).
- \d{2}: Matches exactly two digits (the month and the day parts).
- : Matches a space (the separator between the date and time).
- \d{2}: Matches exactly two digits (the hour part).
- :: Matches a colon (the separator between hours, minutes, and seconds).
- \d{2}: Matches exactly two digits (the minute part).
- :: Matches a colon (the separator between minutes and seconds).
- \d{2}: Matches exactly two digits (the second part).

b. Extract the complete e-mail address from each line.

```
import re
text = """
Contact us at email@example.com for inquiries.
Send your message to john.doe@example.net.
"""
email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b'
emails = re.findall(email_pattern, text)
print(emails)
```

Output:

```
['email@example.com', 'john.doe@example.net']
```

Explanations:

Your regular expression pattern r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b' is designed to match email addresses. Here is a breakdown of the pattern:

- \b: Word boundary, ensures that the match is at the start and end of a word.
- [A-Za-z0-9._%+-]+: Matches one or more characters that can be letters (uppercase or lowercase), digits, dots (.), underscores (_), percent signs (%), plus signs (+), or hyphens (-). This part matches the local part of the email address.
- @: Matches the at symbol (@).
- [A-Za-z0-9.-]+: Matches one or more characters that can be letters (uppercase or lowercase), digits, dots (.), or hyphens (-). This part matches the domain name.
- \.: Matches a literal dot (.) before the domain suffix.
- [A-Za-z]{2,}: Matches two or more letters (uppercase or lowercase), representing the domain suffix (like .com, .org, .net).
- \b: Word boundary, ensures that the match is at the start and end of a word.

c. Extract only the months from the timestamps.

```
import re

# Sample list of timestamps
timestamps = [
'2023-06-25 14:30:45',
'1999-12-31 23:59:59',
'2024-01-01 00:00:00' ]
```

```
# Regular expression pattern to capture the month
month_pattern = r'\d{4}-(\d{2})-\d{2} \d{2}:\d{2}:\d{2}'
# Extract months
months = [re.search(month_pattern, timestamp).group(1) for timestamp in timestamps]

# Print the extracted months
print(months)
```

Output :

```
06
12
01
```

Explanations:

re.search(month_pattern, timestamp).group(1) finds the first match of the pattern in each timestamp and retrieves the month using .group(1), which refers to the first capturing group.

d. Extract only the years from the timestamps.

```
import re
# Sample list of timestamps

timestamps = [
'2023-06-25 14:30:45',
'1999-12-31 23:59:59',
'2024-01-01 00:00:00' ]


# Regular expression pattern to capture the year
year_pattern = r'(\d{4})-\d{2}-\d{2} \d{2}:\d{2}:\d{2}'

# Extract years
years = [re.search(year_pattern, timestamp).group(1) for timestamp in timestamps]

# Print the extracted years
print(years)
```

Output :

```
2023
1999
2024
```

e. Extract only the time (HH:MM:SS) from the timestamps

```
import re

# Sample list of timestamps
timestamps = [
'2023-06-25 14:30:45',
'1999-12-31 23:59:59',
'2024-01-01 00:00:00' ]

# Regular expression pattern to capture the time
time_pattern = r'\d{4}-\d{2}-\d{2} (\d{2}:\d{2}:\d{2})'

# Extract times
times = [re.search(time_pattern, timestamp).group(1) for timestamp in timestamps]

# Print the extracted times
print(times)
```

Output :

```
['14:30:45', '23:59:59', '00:00:00']
```

Practical : 3.

Create utility script to process telephone numbers such that

a. Area codes (the first set of three-digits and the accompanying hyphen) are optional, that is, your regex should match both 800-555-1212 as well as just 555-1212.
b. Either parenthesized or hyphenated area codes are supported, not to mention optional; make your regex match 800-555-1212, 555-1212, and also (800) 555-1212.

```
import re

def process_telephone_number(number):

# Define the regular expression pattern
pattern = r'^\(?(\d{3})\)?[-\s]?(\d{3})[-\s]?(\d{4})$'

# Use re.match to apply the pattern to the number
match = re.search(pattern, number)

if match:
```

```
area_code = match.group(1)
first_three = match.group(2)
last_four = match.group(3)

# Check if area code is provided or not
if area_code:
formatted_number = f'{area_code}-{first_three}-{last_four}'
else:
formatted_number = f'{first_three}-{last_four}'
return formatted_number
else:
return None

# Example usage:
numbers = ["800-555-1212", "555-1212", "(800) 555-1212"]
for number in numbers:
formatted_number = process_telephone_number(number)
if formatted_number:
print(f'Original: {number} -> Formatted: {formatted_number}')
else:
print(f'Invalid format: {number}')
```

Output:

```
Original: 800-555-1212 -> Formatted: 800-555-1212
Invalid format: 555-1212
Original: (800) 555-1212 -> Formatted: 800-555-1212
```