In constructing the regular expressions for detecting phone numbers and dates, we use principles from formal language theory, specifically the design of regular languages and finite automata. Each regular expression was crafted to capture distinct patterns within the text, adhering to constraints specified in the assignment.

Phone Number Detection

The phone number grammar was defined to detect three common formats. Each format corresponds to a specific regular language over the alphabet of digits `0-9`, the hyphen `-`, and parentheses `()`.

1. **Pattern: `999-999-9999`**
    - **Expression**: `\b\d{3}-\d{3}-\d{4}\b`
    - **Explanation**:
        - `\d{3}` denotes exactly three occurrences of any digit, representing the area code and prefix.
        - `-` is a literal character that matches the hyphen separator.
        - `\d{4}` represents exactly four occurrences of any digit, capturing the line number.
        - `\b` denotes word boundaries to ensure that we detect standalone phone numbers, preventing partial matches within larger strings.
2. **Pattern: `9999999999`**
    - **Expression**: `\b\d{10}\b`
    - **Explanation**:
        - `\d{10}` matches exactly ten consecutive digits, representing a phone number without separators.
        - The use of `\b` word boundaries ensures that only standalone ten-digit numbers are matched, ignoring digits within longer sequences.
3. **Pattern: `(999) 999-9999`**
    - **Expression**: `\(\d{3}\) \d{3}-\d{4}`
    - **Explanation**:
        - `\(\d{3}\)` matches an area code enclosed in parentheses. `\(` and `\)` are escape sequences for literal parentheses.
        - A space character separates the area code from the prefix.
        - `\d{3}-\d{4}` represents the remaining phone number segments, where `\d{3}` is the prefix and `\d{4}`is the line number.

These patterns are combined using the `|` (OR) operator, resulting in the complete phone number grammar:

```
phone_pattern = r'\b\d{3}-\d{3}-\d{4}\b|\b\d{10}\b|\(\d{3}\) \d{3}-\d{4}'
```

Date Detection

The date grammar was constructed to capture four different date formats, each with specific constraints on year, month, and day values. These constraints are defined based on formal languages and regular expressions that enforce exact matching for years within `2000-2030` and full month names.

1. **Pattern: `YYYY-MM-DD` (e.g., `2020-02-02`)**
    - **Expression**:
      `\b(20[0-2][0-9])-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])\b`
    - **Explanation**:
        - `(20[0-2][0-9])` restricts the year to the range `2000-2030`. `20[0-2]` captures the first three digits, while `[0-9]` captures the last digit.
        - `(0[1-9]|1[0-2])` restricts the month to values `01-12`, where `0[1-9]` matches `01-09` and `1[0-2]`matches `10-12`.
        - `(0[1-9]|[12][0-9]|3[01])` restricts the day to `01-31`.
2. **Pattern: `YYYY/MM/DD` (e.g., `2020/02/02`)**
    - **Expression**:
      `\b(20[0-2][0-9])/(0[1-9]|1[0-2])/(0[1-9]|[12][0-9]|3[01])\b`
    - **Explanation**: This expression is similar to the `YYYY-MM-DD` format but uses `/` instead of `-` as the separator.
3. **Pattern: `Day, DD Month YYYY` (e.g., `Monday, 8 September 2020`)**
    - **Expression**:
      `\b(Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday),\s(0?[1-9]|[12][0-9]|3[01])\s(January|February|March|April|May|June|July|August|September|October|November|December)\s(20[0-2][0-9])\b`
    - **Explanation**:
        - `(Monday|Tuesday|...|Sunday)` specifies valid day names, using alternation to allow any day of the week.
        - `(0?[1-9]|[12][0-9]|3[01])` matches day numbers from 1 to 31, allowing an optional leading zero.
        - `(January|February|...|December)` specifies valid month names in full, allowing only these exact spellings.
        - `(20[0-2][0-9])` restricts the year to `2000-2030`.
4. **Pattern: `Month DD, YYYY` (e.g., `April 9, 2020`)**
    - **Expression**:
      `\b(January|February|March|April|May|June|July|August|September|October|November|December)\s(0?[1-9]|[12][0-9]|3[01]),\s(20[0-2][0-9])\b`
    - **Explanation**:
        - `(January|February|...|December)` matches any month name in full.
        - `(0?[1-9]|[12][0-9]|3[01])` restricts the day to valid values from 1 to 31.
        - `(20[0-2][0-9])` restricts the year to the range `2000-2030`.

These expressions are combined using the `|` operator to allow matching any of the specified formats:

```
date_pattern = (
```

```
    r'\b(20[0-2][0-9])-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])\b|'   # YYYY-MM-DD
    r'\b(20[0-2][0-9])/(0[1-9]|1[0-2])/(0[1-9]|[12][0-9]|3[01])\b|'   # YYYY/MM/DD

r'\b(Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday),\s(0?[1-9]|[12][0-9]|3[01])\s'

r'(January|February|March|April|May|June|July|August|September|October|November|December)\s(20[0-2][0-9])\b|'  # Day, DD Month YYYY

r'\b(January|February|March|April|May|June|July|August|September|October|November|December)\s'
    r'(0?[1-9]|[12][0-9]|3[01]),\s(20[0-2][0-9])\b'  # Month DD, YYYY
)
```

By constructing these patterns, we ensure that each format adheres to the given requirements, such as valid year ranges, correct month names, and acceptable day names. These regular expressions collectively serve as the grammar for date detection, allowing the program to accurately identify valid date formats within text.

Test Results:

Input: Call me on Monday, April 9, 2020 at 999-999-9999
- Detected phone number: 999-999-9999
- Detected date: April 9 2020

Input: My appointment is scheduled for 2020/02/02, and my phone number is (999) 999-9999
- Detected phone number: (999) 999-9999
- Detected date: 2020 02 02

Input: I was born on 2020-02-02.
- Detected date: 2020 02 02

Input: The meeting is scheduled for Monday, 8 September 2020.
- Detected date: Monday 8 September 2020

Input: Let's meet on April 9, 2020.
- Detected date: April 9 2020