Python

Full stack Skills Bootcamp

Introducing Python Regular Expressions

What are Regular Expression or REGEX?

Regular expressions (regex) are sequences of characters that form search patterns.

They are used for:

- Searching: Locate specific strings or patterns in text.
- Matching: Validate if a string adheres to a specific format (e.g., email, phone numbers).
- Replacing: Modify strings by substituting matches with new values.

- Regular expressions are part of the 're' module in Python, which provides various methods for regex operations.

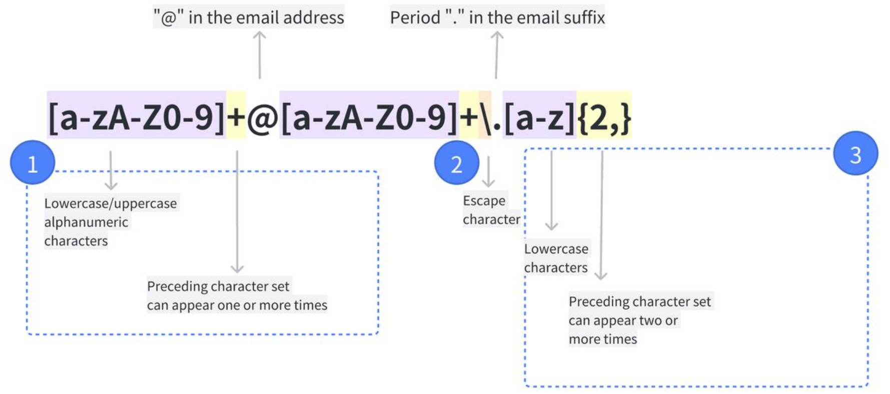
Defining Regex Patterns

Creating Patterns:

- Patterns are defined using special characters and sequences.
- For example: \d matches any digit (0-9), \w matches any alphanumeric character (letters and digits), \s matches any whitespace character (spaces, tabs, newlines).

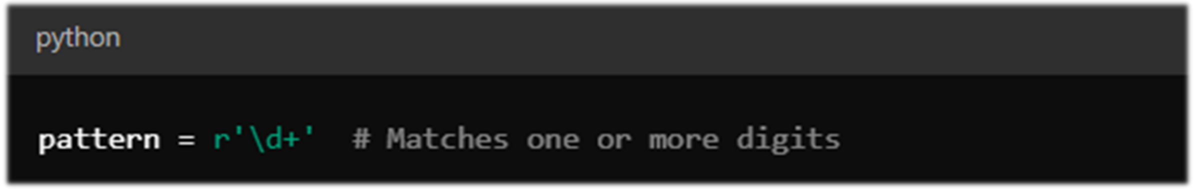
```python
pattern = r'\d+'  # Matches one or more digits
```

- Patterns can be defined as raw strings (using r' ') to avoid escaping backslashes.

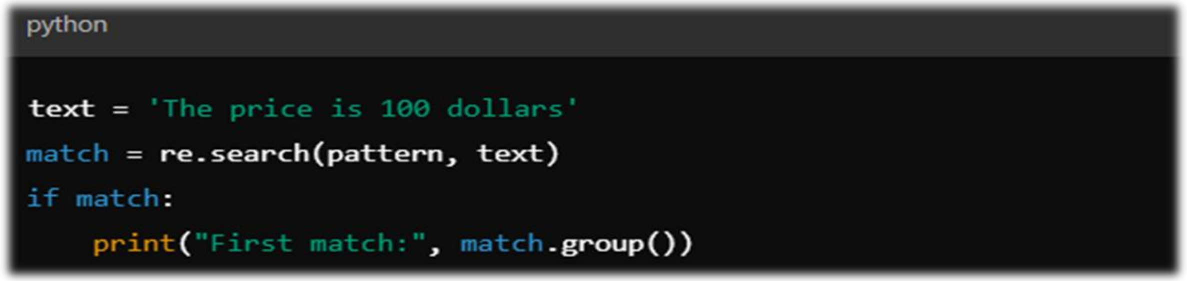

"@" in the email address
Period "." in the email suffix
[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-z]{2,}
1
Lowercase/uppercase alphanumeric characters
Preceding character set can appear one or more times
2
Escape character
3
Lowercase characters
Preceding character set can appear two or more times

Searching for Patterns

- Searching for the First Occurrence
  - Use “.search()” to find the first match of a pattern in a string.



```python
pattern = r'\d+'   # Matches one or more digits
```



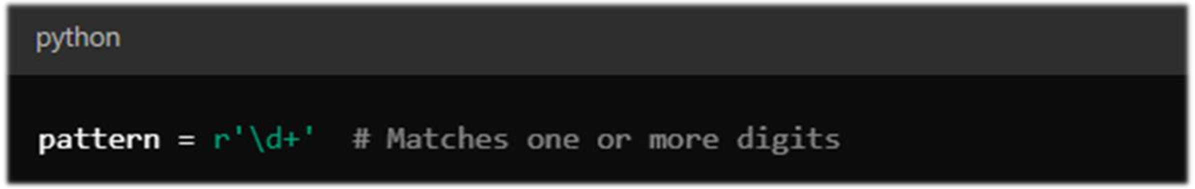
```python
text = 'The price is 100 dollars'
match = re.search(pattern, text)
if match:
    print("First match:", match.group())
```

  - If no match is found, .search() returns None. Useful for checking if a pattern exists without extracting all matches.

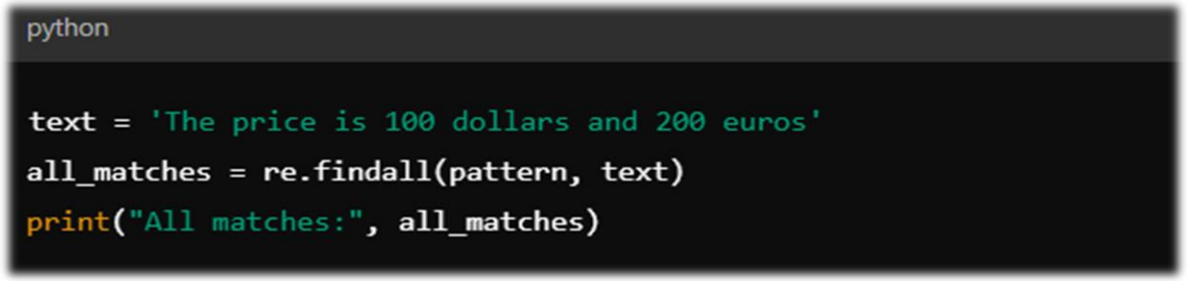
Finding All Matches

Finding All Occurrences

- Use “.findall()” to extract all matches of a pattern from a string.



```python
pattern = r'\d+'  # Matches one or more digits
```



```python
text = 'The price is 100 dollars and 200 euros'
all_matches = re.findall(pattern, text)
print("All matches:", all_matches)
```

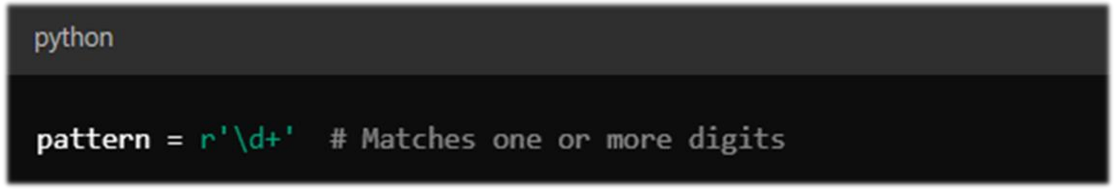
- .search() returns a list of all matches found, allowing easy data extraction for further processing.

Step8Up Academy

Replacing Text

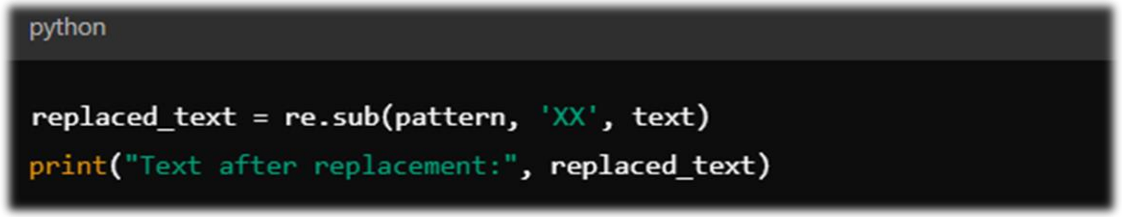
Replacing All Occurrences

- Use “.sub()” to replace all matches of a pattern with a specified string.



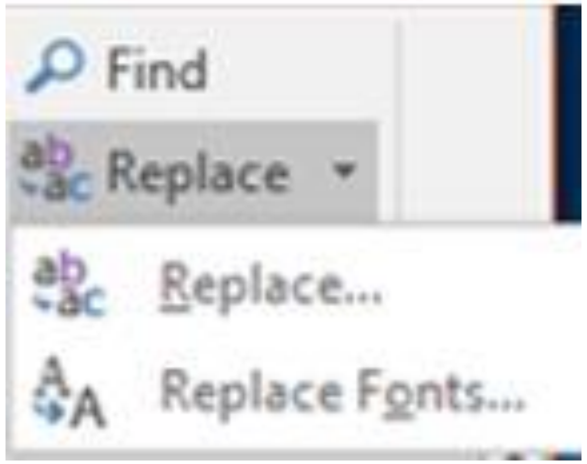
python

```python
pattern = r'\d+'   # Matches one or more digits
```



python

```python
replaced_text = re.sub(pattern, 'XX', text)
print("Text after replacement:", replaced_text)
```

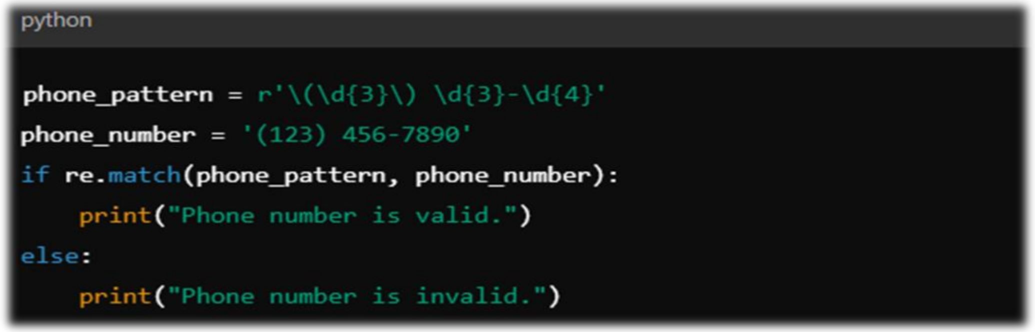


- “.sub()” is helpful for cleaning data or formatting strings before processing or outputting.

Example Usage of Regular Expressions

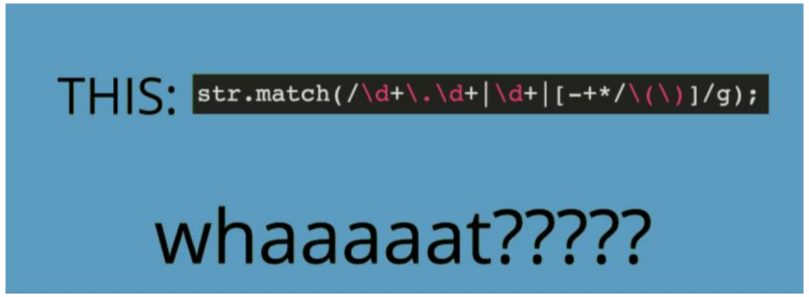
Validating a Phone Number Format

- Regular expressions can ensure user input adheres to specific formats.



python

```python
phone_pattern = r'\(\d{3}\) \d{3}-\d{4}'
phone_number = '(123) 456-7890'
if re.match(phone_pattern, phone_number):
    print("Phone number is valid.")
else:
    print("Phone number is invalid.")
```



THIS: `str.match(/\d+\.\d+|\d+|[-+*/\(\)]/g);`

whaaaaat?????

- This pattern checks for a phone number in the format (XXX) XXX-XXXX.
- Regex is particularly useful in form validations, ensuring users provide data in expected formats.

Step8Up Academy

Conclusion

- Regular expressions are powerful tools for:
  - Text Processing: Efficiently searching, matching, and modifying strings.
  - Pattern Matching: Identifying specific sequences in strings, useful for data validation and extraction.
  - Flexibility: Regex can handle complex string manipulations with concise syntax.

reg[ular]
expr[essio]n

Understanding regex syntax and methods enhances your ability to work with text data in Python and other programming languages