# ## 1. What is the name of the feature responsible for generating Regex objects?

**Ans:-** `compile()` feature is responsible for generating RegEx objects.

**For Example** In below example a pattern object is created using `compile()`

In [22]:
```
import re
my_string = "shri@gmail.com"
pattern = re.compile(r"([a-zA-Z]+)@([a-zA-Z]+)\.([a-zA-Z]+)")

matches = pattern.finditer(my_string)
for match in matches:
    print(match.group(0))
```

```
shri@gmail.com
```

# 2. Why do raw strings often appear in Regex objects?

**Ans:-** RegEx use the backslash (\) for Metacharacters [. ^ $ * + ? { } \ | ( )] or speical sequences [\s, \S, \w, \W, etc]. But in python (\) is also used to escape characters. so to avoid this we use raw strings.

**For example**

In [34]:
```
s = "\tramesh"
print(s)
s1 = r"\tramesh"
print(s1)
```

```
	ramesh
\tramesh
```

# 3. What is the return value of the search() method?

**Ans:-** Return value of a `search()` method is a `match object` or `None` if no match found.

**For Example**

In [35]:
```
my_string = "shubham@gmail.com"
pattern = re.compile(r"([a-zA-Z]+)@([a-zA-Z]+)\.([a-zA-Z]+)")

match = pattern.search(my_string)
print(match)
```

```
<re.Match object; span=(0, 17), match='shubham@gmail.com'>
```

In [36]:
```
my_string = "shubham@gmail.com"
pattern = re.compile(r"^rty")

match = pattern.search(my_string)
print(match)
```

```
None
```

## 4. From a Match item, how do you get the actual strings that match the pattern?

**Ans:-** `group()` method is used to get he actual strings that match object.

**For Example**

In [37]:
```
my_string = "shubham@1234"
pattern = re.compile(r"@\d+")

match = pattern.search(my_string)
print(match.group(0))
```

```
@1234
```

## 5. In the regex which created from the r'(\d\d\d)-(\d\d\d-\d\d\d\d)', what does group zero cover? Group 2? Group 1?

**Ans:-**

1. `group(0)` covers entire pattern i.e `(\d\d\d)-(\d\d\d-\d\d\d\d)`
2. `group(1)` covers first part i.e. `(\d\d\d)`
3. `group(2)` covers second part i.e. `(\d\d\d-\d\d\d\d)`

**For Example**

In [38]:
```
my_str = """
999-345-3456
1234-123-345
123-4565-456
"""
pattern = re.compile(r"(\d\d\d)-(\d\d\d-\d\d\d\d)")

matches = pattern.finditer(my_str)

for match in matches:
    print("group zero:\t",match.group(0))
    print("group one:\t",match.group(1))
    print("group two:\t",match.group(2))
```

```
group zero:	 999-345-3456
group one:	 999
group two:	 345-3456
```

## 6. In standard expression syntax, parentheses and intervals have distinct meanings. How can you tell a regex that you want it to fit real parentheses and periods?

**Ans:-** `'\'` along with parenthesis and period will be used to fit real parentheses and periods in a RegEx pattern.

**For Example** `'\('` , `'\)'` and `'\.'`

In [39]:
```
my_str = """
999-(345)-3456.
1234-123-345
123-4565-456
"""
pattern = re.compile(r"(\d\d\d)-(\(\d\d\d\)-\d\d\d\d)\.")

matches = pattern.finditer(my_str)

for match in matches:
    print(match.group(0))
```

```
999-(345)-3456.
```

## 7. The findall() method returns a string list or a list of string tuples. What causes it to return one of the two options?

**Ans** If RegEx object has no groups then a list containing matching strings is returned. whereas if group is present then a list containing tuple of strings is returned.

**For Example**

In [40]:
```
my_str = """
999-345-3456
"""
pattern1 = re.compile(r"(\d\d\d)-(\d\d\d-\d\d\d\d)")
match1 = pattern1.findall(my_str)
print(match1)

pattern2 = re.compile(r"\d\d\d-\d\d\d-\d\d\d\d")
match2 = pattern2.findall(my_str)
print(match2)
```

```
[('999', '345-3456')]
['999-345-3456']
```

## 8. In standard expressions, what does the | character mean?

**Ans:-** | character means `or` in standard expresion.

## 9. In regular expressions, what does the character stand for?

**Ans:-** `?` stands for `"zero or one occurances and is used when a character can be optional"` in regular expression.

**For Example** In below example `?` is used to make `.` optional

In [41]:
```
my_string = """
Mr. Kumar
Mrs Kumar
Mr Abhishek
Ms. jeetisha
"""

pattern = re.compile(r"(Mr|Ms|Mrs)\.?\s\w+")
matches = pattern.finditer(my_string)

for match in matches:
    print(match)
```

```
<re.Match object; span=(1, 10), match='Mr. Kumar'>
<re.Match object; span=(11, 20), match='Mrs Kumar'>
<re.Match object; span=(21, 32), match='Mr Abhishek'>
<re.Match object; span=(33, 45), match='Ms. jeetisha'>
```

## 10. In regular expressions, what is the difference between the + and * characters?

**Ans:-** `+` and `*` are both quantifiers

1. `+` indicates one or more occurances of a preceding group.
2. `*` indicates zero or more occurances of a preceding group.

**For Example**

In [42]:
```
my_string = "Wonderman"

patt1 = re.compile(r"Wonder(wo)+man")

match1 = patt1.search(my_string)
print(match1)

patt2 = re.compile(r"Wonder(wo)*man")

match2 = patt2.search(my_string)
print(match2)
```

```
None
<re.Match object; span=(0, 9), match='Wonderman'>
```

## 11. What is the difference between {4} and {4,5} in regular expression?

**Ans:-**

1. `{4}` This means that the preceding character or group should occur 4 times.
2. `{4,5}` This means that the preceding character or group should occur minimum 4 times and maximum 5 times.

**For Example**

In [43]:
```
string1 = "2022-08-26"

pattern1 = re.compile(r"\d{4}-\d{1,3}-\d{2}")

match = pattern1.findall(string1)
print(match)
```

```
['2022-08-26']
```

## 12. What do you mean by the \d, \w, and \s shorthand character classes signify in regular expressions?

**Ans:**

1. `\d` : Matches any digit; this is equivalent to the class [0-9].
2. `\s` : Matches any whitespace character;
3. `\w` : Matches any alphanumeric (word) character; this is equivalent to the class [a-zA-Z0-9_].

## 13. What do means by \D, \W, and \S shorthand character classes signify in regular expressions?

**Ans:-**

1. `\D` : Matches any non-digit character; this is equivalent to the class [^0-9].
2. `\S` : Matches any non-whitespace character;
3. `\W` : Matches any non-alphanumeric character; this is equivalent to the class [^a-zA-Z0-9_].

## 14. What is the difference between *.? and .?*

***Ans:-** `.*?` is a non greedy mode which returns the shortest string that matches the pattern. Whereas `.*` is a Greedy mode, which returns the longest string that matches the pattern.

## 15. What is the syntax for matching both numbers and lowercase letters with a character class?

**Ans:-** `re.compile(r"[a-z0-9]")` or `re.compile(r"[0-9a-z]")` is the syntax for matching both numbers and lowercase letters with a character class

## 16. What is the procedure for making a Ans:- `IGNORECASE` (`re.IGNORECASE` or `re.I`) flag can be used to make normal expression in regax case insensitive.

**For Example**

In [44]:
```
string = "ShUbHaM"

pattern = re.compile(r"shubham", re.I)

match = pattern.findall(string)
print(match)
```

```
['ShUbHaM']
```

## 17. What does the . character normally match? What does it match if re.DOTALL is passed as 2nd argument in re.compile()?

**Ans:-**

1. `.` character matches any character except newline character.
2. `re.DOTALL` is a flag that is used to match any character including newline character.

## 18. If numReg = re.compile(r'\d+'), what will numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen') return?

**Ans:-** It will return `'X drummers, X pipers, X rings, X hen'` **Below is the execution of code**

In [45]:
```
numReg = re.compile(r'\d+')
numReg.sub('X', '11 drummers, 10 pipers, five rings, 4 hen')
```

Out[45]:
```
'X drummers, X pipers, five rings, X hen'
```

## 19. What does passing re.VERBOSE as the 2nd argument to re.compile() allow to do?

**Ans:-** `re.VERBOSE` or `re.X` flag enables us to pass pattern as multiline comment and include whitespace.

**For Example**

In [46]:
```
emails = """
pythonengineer@gmail.com
Python-engineer@gmx.de
python-engineer123@my-domain.org
"""
# return all emails
```

In [47]:
```
# without VERBOSE

pattern = re.compile(r"[a-zA-z0-9-]+@[a-zA-Z-]+\.(com|de|org)")
matches = pattern.finditer(emails)

for match in matches:
    print(match.group(0))
```

```
pythonengineer@gmail.com
Python-engineer@gmx.de
python-engineer123@my-domain.org
```

In [48]:
```
# with VERBOSE

pattern = re.compile(r"""[a-zA-z0-9-]+
                         @
                         [a-zA-Z-]+
                         \.
                         (com|de|org)
                         """, re.X)
matches = pattern.finditer(emails)

for match in matches:
    print(match.group(0))
```

```
pythonengineer@gmail.com
Python-engineer@gmx.de
python-engineer123@my-domain.org
```

## 20. How would you write a regex that match a number with comma for every three digits? It must match the given following:

'42'

'1,234'

'6,368,745'

but not the following:

'12,34,567' (which has only two digits between the commas)

'1234' (which lacks commas)

**Ans:-**

In [49]:
```
l =  ['42', '1,234', '6,368,745', '12,34,567', '1234']

pattern = re.compile(r"^\d{1,2}(,\d{3})*$")
for i in l:
    match = pattern.search(i)
    print(match)
```

```
<re.Match object; span=(0, 2), match='42'>
<re.Match object; span=(0, 5), match='1,234'>
<re.Match object; span=(0, 9), match='6,368,745'>
None
None
```

## 21. How would you write a regex that matches the full name of someone whose last name is Watanabe? You can assume that the first name that comes before it will

## always be one word that begins with a capital letter. The regex must match the following:

'Haruto Watanabe'

'Alice Watanabe'

'RoboCop Watanabe'

but not the following:

'haruto Watanabe' (where the first name is not capitalized)

'Mr. Watanabe' (where the preceding word has a nonletter character)

'Watanabe' (which has no first name)

'Haruto watanabe' (where Watanabe is not capitalized)

**Ans:**

In [18]:
```
names = ['Haruto Watanabe', 'Alice Watanabe', 'RoboCop Watanabe','haruto Watan
         'Watanabe', 'Haruto watanabe']

pattern = re.compile(r"([A-Z]{1}[a-z]*|[A-Z]{1}[a-z]*[A-Z]{1}[a-z]*)\sW{1}[a-z

for name in names:
    match = pattern.search(name)
    print(match)
```

```
<re.Match object; span=(0, 15), match='Haruto Watanabe'>
<re.Match object; span=(0, 14), match='Alice Watanabe'>
<re.Match object; span=(0, 16), match='RoboCop Watanabe'>
None
None
None
None
```

## 22. How would you write a regex that matches a sentence where the first word is either Alice, Bob, or Carol; the second word is either eats, pets, or throws; the third word is apples, cats, or baseballs; and the sentence ends with a period? This regex should be case-insensitive. It must match the following:

'Alice eats apples.'

'Bob pets cats.'

'Carol throws baseballs.'

'Alice throws Apples.'

'BOB EATS CATS.'

but not the following:

'RoboCop eats apples.'

'ALICE THROWS FOOTBALLS.'

'Carol eats 7 cats.'

**Ans:**

In [19]:
```
sentences = ['Alice eats apples.','Bob pets cats.','Carol throws baseballs.','
             'BOB EATS CATS.','RoboCop eats apples.','ALICE THROWS FOOTBALLS.',

pattern = re.compile(r"(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|bas

for i in sentences:
    match = pattern.search(i)
    print(match)
```

```
<re.Match object; span=(0, 18), match='Alice eats apples.'>
<re.Match object; span=(0, 14), match='Bob pets cats.'>
<re.Match object; span=(0, 23), match='Carol throws baseballs.'>
<re.Match object; span=(0, 20), match='Alice throws Apples.'>
<re.Match object; span=(0, 14), match='BOB EATS CATS.'>
None
None
None
```

In [ ]:

In [ ]: