5. Regular Expression

General Guideline

Topics Covered

Day 1	Day 2	Day 3	
5.1 GETTING FAMILIAR WITH REGULAR EXPRESSION 5.1.1 INTRODUCTION TO REGEX 5.1.2 REGULAR EXPRESSION IN PYTHON.	5.2 HOW TO CREATE PATTERN	5.3 REGEX ESSENTIAL METHODS 5.3.1 SEARCH () METHOD 5.3.2 MATCH () METHOD 5.3.3 FINDALL () METHOD 5.3.4 FLAG ARGUMENT IN REGEX METHOD	

Session Plan - Day 1

5.1 GETTING FAMILIAR WITH REGULAR EXPRESSION

5.1.1 INTRODUCTION TO REGEX

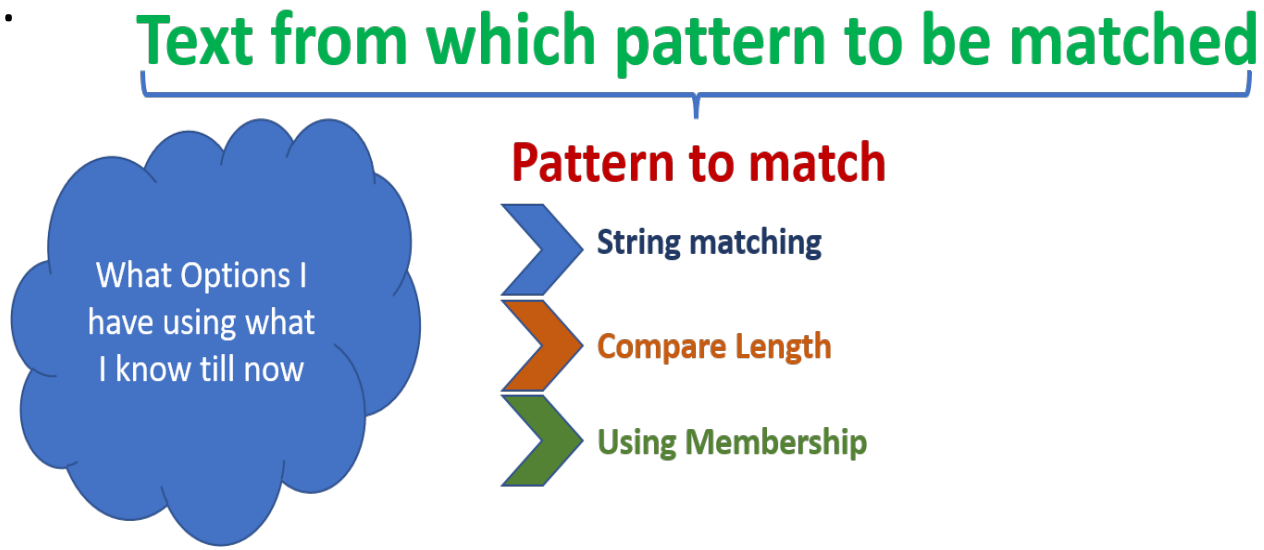
5.1.2 REGULAR EXPRESSION IN PYTHON.

Regular Expression

RegEx

Without Regex

- I want that each student password should:
  - Start with letter and end with a number
  - It should be of length 8 and
  - It should contain at least 1 special character.



- Solutions using what we have learnt:
  - Check each condition using string matching one by one.
  - Compare length of each password using len() function
  - Specifically check for first and last position using membership

Introduction to Regex

- Regular expression is also known as regexes or regex.
- A regex is a character series that specifies a pattern for string matching.
- Text can be searched, edited, and manipulated using regex.

Raw Python strings

- Raw strings are raw string literals that treat backslash (\ ) as a literal character.
- It is useful when we want to have a string that contains backslash(\) and don't want it to be treated as an escape character.
- For example, if we try to print a string with a "\n" inside, it will add one line break.
- But if we mark it as a raw string, it will simply print out the "\n" as a normal character.
- Python raw strings are prefixed with 'r' or 'R'. Prefix a string with 'R' or 'r' and it will be treated as a raw string.

```
raw_s = r'Hi\nHello'
print(raw_s)
```

Raw Python strings

- Raw strings are raw string literals that treat backslash (\ ) as a literal character.
- It is useful when we want to have a string that contains backslash(\) and don't want it to be treated as an escape character.
- For example, if we try to print a string with a "\n" inside, it will add one line break.
- But if we mark it as a raw string, it will simply print out the "\n" as a normal character.
- Python raw strings are prefixed with 'r' or 'R'. Prefix a string with 'R' or 'r' and it will be treated as a raw string.

Raw Python strings

```
text= "This is a \n normal string"
print(text)

raw_text = r"This is a \n raw string"
print(raw_text)
```

'r' treats \ as a literal character

```
This is a
 normal string
This is a \n raw string
```

Output

- we can see that the first string text includes one new-line and the second raw string also include one new-line character.
- But the new line was printed as '\n' for the second string.

Regular Expression in Python

- To use Regex in python program we need to import a module re
- The re module in Python contains the regex functions.
- Many useful functions and methods are available in the re module.

Name of the module

```
import re
```

keyword

Can you answer these questions?

1. Which module in Python supports regular expressions?

A) re

B) regex

C) reregex

D) piregex

Can you answer these questions?

2. Using regex we can

A) Search

B) Edit

C) Modify

D) All of the above

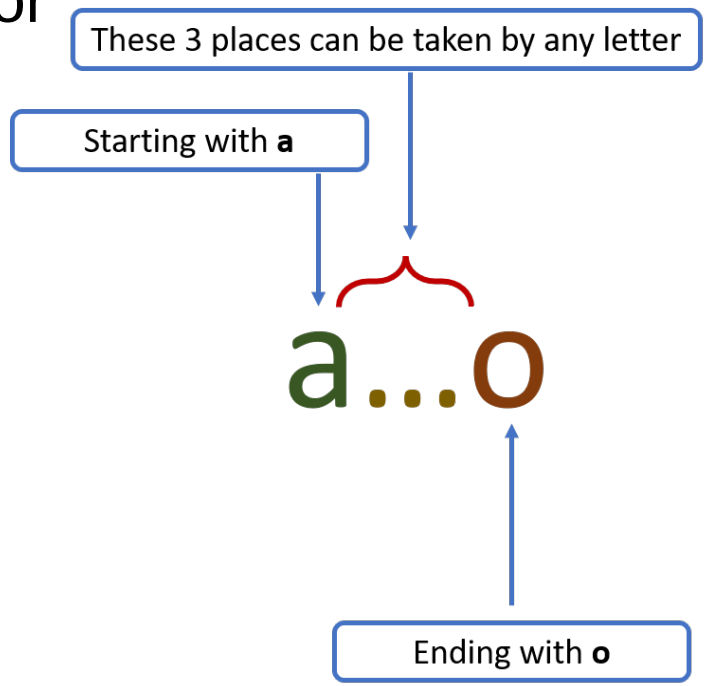
Session Plan - Day 2

5.2 HOW TO CREATE PATTERN

Meta Characters

How to create pattern

- Let's understand suppose I want to search my text book for a word which start with letter a and ending with o and having length of 5.
- Now, if we have understood that in this example, we have specified a pattern and words like 'aggro','amino','anglo' match our pattern.



Meta Characters

Now real **question** is how we would define a pattern in a python program.

To **answer** this, we have

Meta Characters.

Meta Characters

- Meta characters are characters which contain a special meaning when they are read in regular expression environment

[] . ^ $ * + ? {} () \ |

Meta Characters

- Square Brackets ([ ])
- Whatever character we would write inside square bracket, that character would be matched.

Pattern	String	Matched or Not
[abcd]	a	1 match
	b	1 match
	c	1 match
	d	1 match
	ab	2 match
	abd	3 match
	ab cd ef	4 match
	efgh i	No match

Meta Characters

- We can also define interval inside square bracket like [a-d] is same as [abcd], [0-9] is same as [0123456789].
- If we put ^ (caret symbol) as a prefix of pattern inside square bracket, then it reverses the meaning of pattern.
- For example:
  - [^de] means any character except d and e.
  - [^1-3] means any number except 1,2 and 3.

Meta Characters

- Period or dot(.)
- Period matches single character in the string. In the given example there are two dots it means that it would match 2 consecutive characters.

Pattern	String	Matched or Not
..	a	No match
	b	No match
	c	No match
	d	No match
	ab	1 match
	abde	2 match

Meta Characters

- Carot (^)
- If we want to check whether string is starting with a mentioned character.
- For example, if I want to check if a string starting with letter Z, then pattern I would create is “^Z”.

Pattern	String	Matched or Not
^a	a b ab abc ab abde	1 match No match 1 match 1 match 1 match 1 match
^cd	c d cd ced	No match No match 1 match No match

Meta Characters

- Dollar Sign ($)
- If we have to check the ending character of string then we use dollar.
- For example,

Pattern	String	Matched or Not
b$	b cab ball	1 match 1 match No match

Meta Characters

- Star(*)
- Star check if string has 0 or more occurrence.
- For example,

Pattern	String	Matched or Not
am*b	ab amb ammb amin	1 match 1 match 1 match No match

Meta Characters

- Plus(+)
- Plus check if string has 1 or more occurrence.
- For example,

Pattern	String	Matched or Not
a+b	b ab aab	No match 1 match 1 match

Meta Characters

- Question Mark(?)
- Question mark check if string has 0 or 1 occurrence.
- For example,

Pattern	String	Matched or Not
a?b	b ab aba	1 match 1 match 1 match

Meta Characters

- Braces({})
- Braces are used when we want to specify at least and at most repetitions
- For example a{2,5} means at least 2 occurrences of a and maximum 5 occurrences of a.

Pattern	String	Matched or Not
a{2,5}	a aa abca aaaaabaa	No match 1 match No match 2 match

Meta Characters

- Backslash(\)
- We can have some requirements where we want to use metacharacter symbol in their normal form,
- It means we want regular expression environment to treat them as a symbol not as a metacharacter.
- In this example $ is dollar not metacharacter.

Escape Metacharacter making $ as normal Symbol

\$b

Not a metacharacter, just normal symbol because of $

Meta Characters

- Alternation(|)
- It is used as OR operator for example,

It will search for either a and b in the string

$$\overbrace{a \mid b}$$

Meta Characters

- Group(())
- If we want to join more than one patterns then group metacharacter can be used.
- For example, if I want my string should start with either a or m followed by numeric 1. Pattern created is as shown in figure.

Pattern	String	Matched or Not
(a|m)1	a1 b1 m1 am1	1 match No match 1 match 1 match

Can you answer these questions?

2. Which character stand for Starts with in regex?

A) $

B) ^

C) &

D) #

Can you answer these questions?

2. Which character stand for Zero or more occurrences in regex?

A) #

B) @

C) *

D) |

Session Plan - Day 3

5.3 REGEX ESSENTIAL METHODS

5.3.1 SEARCH () METHOD

5.3.2 MATCH () METHOD

5.3.3 FINDALL () METHOD

5.3.4 FLAG ARGUMENT IN REGEX METHOD

RegEx essential methods

- Now we understand how pattern can be created using metacharacters.
- Re module has many useful functions.
- These functions help us to search and match specified pattern.

Search () method

Name of the module

String in which pattern is to be searched

```
re.search(pattern, string)
```

Method name

Regular expression to be matched

Example

Module Name

```
import re
string="ABES Engineering College"
ma=re.search("AB",string)
print(ma)
if(ma):
    print("Search Successful")
else:
    print("Search UnSuccessful")
```

String where pattern is to be searched

Search method in which we are giving pattern AB to be searched in string

Checking if Search is successful or not

```
<re.Match object; span=(0, 2), match='AB'>
Search Successful
```

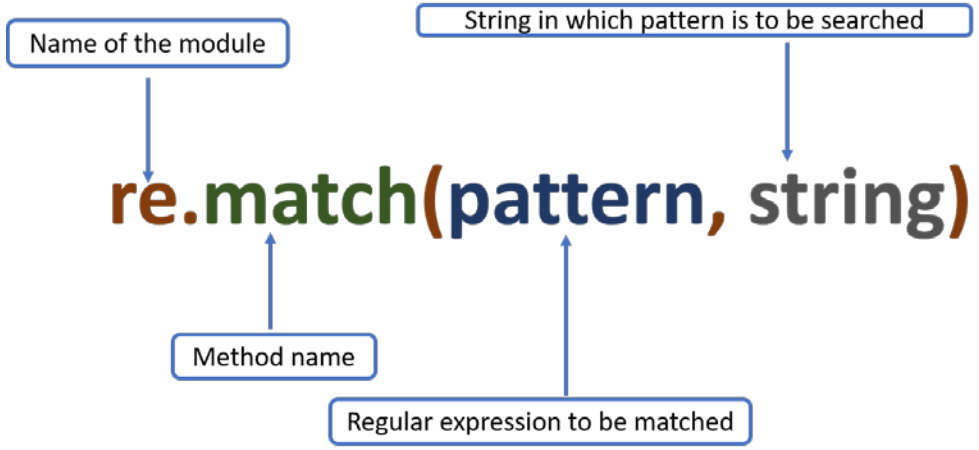
Output

Index at which match is found

Length of pattern which is 2 in this case

Match () method

- Match works same as search method
- But difference is match method only look for the pattern in the beginning while search scans complete string.



Example

String in which pattern to be searched
Test is at the end

```
import re
string="String sample Test"
ma=re.match("Test",string)
print(ma)
if(ma):
    print("Search Successful")
else:
    print("Search UnSuccessful")
```

```
None
Search UnSuccessful
```

Output

String in which pattern to be searched
Test is at the Beginning

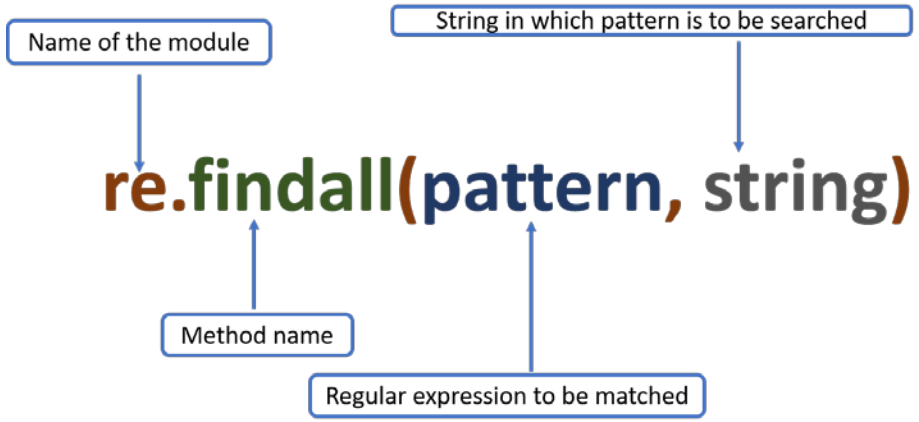
```
import re
string="Test String sample"
ma=re.match("Test",string)
print(ma)
if(ma):
    print("Search Successful")
else:
    print("Search UnSuccessful")
```

Search Successful because pattern is at the beginning

```
<re.Match object; span=(0, 4), match='Test'>
Search Successful
```

Findall () method

- We have seen that search and match method find single occurrence in the string.
- The findall () method is used to find "all" instances of a given pattern in the string.



Example

String in which pattern to be searched
Test is coming two times

```
import re
string="Test String sample Test"
ma=re.findall("Test",string)
print(ma)
if(ma):
    print("Search Successful")
else:
    print("Search UnSuccessful")
```

```
['Test', 'Test']
Search Successful
```

Output

Flag Argument in Regex Method

- Flag is the optional argument in regex methods like search, match and findall.
- Flag is used when we need to modify the standard behavior of regex patterns.

Flag Argument in Regex Method

- For example, lets understand a scenario if I am looking for the occurrences of character “i” in the string and I don’t want to consider the **upper** and **lower** case as a different character so small case i and capital case I both are same as per requirement then I can set this flag as ignore case.

Flag Set to ignore case

```
import re
pattern="i"
string1="Information is immediate"
s1=re.findall(pattern,string1,flags=re.IGNORECASE)
s1
```

```
['I', 'i', 'i', 'i', 'i']
```

Contains small case i and capital case I

Flag Argument in Regex Method

- There are various options for flags as per following table.

Syntax	Long syntax	Meaning
re.I	re.IGNORECASE	Ignore case.
re.M	re.MULTILINE	Make begin/end {^, $} consider each line.
re.S	re.DOTALL	Make . match newline too.
re.U	re.UNICODE	Make {\w, \W, \b, \B} follow Unicode rules.
re.L	re.LOCALE	Make {\w, \W, \b, \B} follow locale.
re.X	re.VERBOSE	Allow comment in regex.

Can you answer these questions?

2. The expression a{5} will match _____ characters with the previous regular expression.

A) 5 or less

B) exactly 5

C) 5 or more

D) exactly 4

Can you answer these questions?

2. In Regex, [a-n] stands for?

A) Returns a match for any digit between 0 and 9

B) Returns a match for any lower case character, alphabetically between a and n

C) Returns a match for any two-digit numbers from 00 and 59

D) Returns a match for any character EXCEPT a, r, and n

Session Plan - 4

5.4 SIGNIFICANCE OF CHARACTER CLASSES & SPECIAL SEQUENCES

5.4.1 CHARACTER CLASSES

5.4.2 SPECIAL SEQUENCES

Character Classes

The character classes are sets of characters or ranges of characters enclosed by square brackets [].

For example,

- **[0-9]** it means match any digit from 0 to 9.
- **[4-8]** it means match any digit from 4 to 8.
- **[A-Z]** it means match any letters from A to Z.

Character Classes

Let's see some of the most common character classes and Its Behavior–

Character Class	Meaning/Role/Behaviour
[pqr]	[pqr]: Match the letter p or q or r
[pqr][st]	[pqr][st]: Match the letter p or q or r followed by either s or t
[^abc]	[^abc]: Match any letter except a,b,or c
[0-9]	[0-9]: Match any digit from 0 to 9
[a-z]	[a-z]: Match any lowercase letters from a to z
[A-Z]	[A-Z]: Match any uppercase letters from A to Z
[a-zA-Z]	[a-zA-Z]: Match any lowercase or uppercase letter
[s-t1-9]	[s-t1-9]: Match the letter between s and t and digites from 1 to 9 but not t1
[a-zA-Z0-9_]	[a-zA-Z0-9_]: Match any alphanumeric character

Character Classes - Example

Example – Using regular expression, check if email id is in correct format or not

Solution – Email - Id has been broken in three parts

XYZ @ abes.ac.in

before @ part, after @part and after dot(.) part

Character Classes - Example

```
import re
pattern="@"
string1="a@a.com"
string2="a.com"
s1=re.findall(pattern,string1)
print(s1)
```

```
['@']
```

Output

Searching for @

```
import re
pattern="\w+@\w\.\w+"
string1="a@a.com"
s1=re.findall(pattern,string1)
print(s1)
```

```
['a@a.com']
```

Searching for any character before @, after @ and after dot(.)

```
import re
pattern="@\w\."
string1="a@a.com"
s1=re.findall(pattern,string1)
print(s1)
```

```
['@a.']
```

Output

Searching for @ followed by any character or digit and dot(.)

Character Classes - Example

Pattern to specify email-id

```
patternemail="[A-Za-z0-9\.\+_-]+@[A-Za-z0-9\._-]+\.[a-zA-Z]*$"
test="anurag.mishra@abes.ac.in"
test1="anurag.mishra.com"
ab=re.match(patternemail,test)
ab1=re.match(patternemail,test1)
print(ab)
print(ab1)
```

First email-id to check if valid or not

Second email-id to check if valid or not

```
<re.Match object; span=(0, 24), match='anurag.mishra@abes.ac.in'>
None
```

Output

First email-id valid

Second email-id not valid

Can you answer these questions?

What will be the output of the given code?

```
import re

re.findall('[aeiou]','i like python')
```

a) ['i', 'i', 'e', 'o']

b) ['i', 'e', 'o']

c) 4

d) None

Can you answer these questions?

Complete the given program to check if a string has at least 1 zero.

a) '0'

b) '[0] '

c) '[0-9]'

d) None

```
import re
pattern = '__________'
if re.search(pattern,'192 Patel nagar'):
    print("string has at least 1 zero")
else:
    print("string has No zero")
```

Special Sequences

The **special sequence** represents the basic predefined character classes, which have a unique meaning.

They are written as **a backslash ** followed by **any character** and it has a special meaning.

For example,

- **\d** sequence is similar to character class [0-9] , which means match any digit from 0 to 9.
- \w sequence is similar to [a-zA-Z0-9_], which means match any lowercase, uppercase, digit and underscore.

Special Sequences

Some of the special sequences are given in the figure below

Special Sequences	Meaning/Role/Behaviour
\A	\A: Matches the specified characters at the beginning of the string
\Z	\Z: Matches the specified characters are at the end of the string
\d	\d: Matches the string contains digits
\D	\D: Matches the string does not contain digits
\s	\s: Matches the string contains a white space character
\S	\S: Matches the string doen not contains a white space character
\w	\w: Matches any characters from a to Z, digits from 0-9, and the" _"
\W	\W: Matches the any characters not from [a to Z, digits from 0-9, and the " _"]
\b	\b: Matches the specified characters are at the beginning or at the end
\B	\B: Matches the specified characters are present, but NOT at the start or at the end

Special Sequence \A

The \A sequences only match the beginning of the string. It works the same as the carrot (^) metacharacter.

Note: If we do have a multi-line string, then **\A** will still match only at the beginning of the string, while the carrot (^) will match at the beginning of each new line of the string.

Special Sequence \A

Example –

```
import re

txt = "The ABESEC in Ghaziabad"

#Check if the string starts with "The":

x = re.findall("\AThe", txt)
y = re.findall("\A ABESEC", txt)

print(x)
print(y)
```

Pattern without Space

Pattern with Space

```
['The']
[]
```

Output

Special Sequence \Z

Backslash Z (\Z) sequences only match the end of the string.

It works the same as the dollar ($) meta-character.

```
import re

txt = "ABESEC in Ghaziabad"

#Check if the string ends with "Ghaziabad":

x = re.findall("Ghaziabad\Z", txt)

y = re.findall("ABESEC\Z", txt)
z = re.findall("Ghaziabad \Z", txt)

print(x)
print(y)
print(z)
```

Pattern without Space

Different Pattern With \Z

Pattern with Space

```
['Ghaziabad']
[]
[]
```

Output

Special Sequence \d

Backslash d or \d matches any digits from 0 to 9 inside the target string.

This special sequence is to character class [0-9] .

```
import re

txt = "ABES 19th KM Stone, NH 24, Ghaziabad"

#Check if the string contains any digits (from 0-9):

x = re.findall("\d", txt)

print(x)
```

Special Sequence

```
['1', '9', '2', '4']
```

Output

Special Sequence \D

Backslash D or \D is the exact opposite of \d. Any character in the target string that is not a digit would be the equivalent of the \D.

Also, we can write \D using **character class [^0-9].**

```
import re

txt = "ABES NH 24"

#Return a match at every no-digit character:

x = re.findall("\D", txt)

print(x)
```

Special Sequence

```
['A', 'B', 'E', 'S', ' ', 'N', 'H', ' ']
```

Output

Special Sequence \w

Backslash w or \w matches any alphanumeric character, also called a word character. This includes lowercase and uppercase letters, the digits 0 to 9, and the underscore character which is Equivalent to character **class [a-zA-z0-9_].**

```
import re

txt = "ABESEC NH_24"

#Return characters from a to Z, digits from 0-9 and _

x = re.findall("\w", txt)

y = re.findall("[a-zA-z0-9_]", txt)
print(x)
print(y)
```

Special Sequence

Character Class

```
['A', 'B', 'E', 'S', 'E', 'C', 'N', 'H', '_', '2', '4']
['A', 'B', 'E', 'S', 'E', 'C', 'N', 'H', '_', '2', '4']
```

Output

Special Sequence \W

Backslash W or \W is the exact opposite of \w, i.e., It matches any NON-alphanumeric character. **\W** same as character class **[^a-zA-z0-9_]** .

```
import re

txt = "ABESEC NH_24!"

#Returns the string which DOES NOT contain any word characters

x = re.findall("\W", txt)

y = re.findall("[^a-zA-z0-9_]", txt)
print(x)
print(y)
```

Special Sequence

Character Class

```
[' ', '!']
[' ', '!']
```

Output

Special Sequence \s

Backslash s or \s matches any whitespace characters (Spaces, tab character (\t) ,newline character (\n) etc) inside the target string.

```
import re

txt = "ABESEC NH_24 "

#Return a match at every white-space character

x = re.findall("\s", txt)

y = re.findall("[ \t\n\x0b\r\f]", txt)
print(x)
print(y)
```

Special Sequence

Character Class

```
[' ', ' ']
[' ', ' ']
```

Output

Special Sequence \S

Backslash S or \S is the exact opposite of \s, and it matches any character which is not a whitespace character.

```
import re

txt = "ABESEC NH_24 "

#Returns the string DOES NOT contain a white space character

x = re.findall("\S", txt)

y = re.findall("[^  \t\n\x0b\r\f]", txt)
print(x)
print(y)
```

Special Sequence

Character Class

```
['A', 'B', 'E', 'S', 'E', 'C', 'N', 'H', '_', '2', '4']
['A', 'B', 'E', 'S', 'E', 'C', 'N', 'H', '_', '2', '4']
```

Output

Special Sequence \b

Backslash b or \b matches the empty string, but only at the beginning or end of a word.

```
import re

txt = "ABESEC NH_24 or ABESEC Ghzb "

#Check if "ABESEC" is present at the beginning of a WORD
#Check if "EC" is present at the end of a WORD

x = re.findall(r"\bABESEC", txt)

y = re.findall(r"\b ABESEC", txt)

z = re.findall(r"EC\b", txt)

print(x)
print(y)
print(z)
```

Special Sequence for start

Space with Special Sequence

Special Sequence for end

```
['ABESEC', 'ABESEC']
[' ABESEC']
['EC', 'EC']
```

Output

Special Sequence \B

Backslash B or \B is the exact opposite of \b, it matches the empty string, but only when it is not at the beginning or end of a word.

```
import re

txt = "TheABESEC NH_24 is an Engineering College"

#Check if "ABESEC" is present, but NOT at the beginning of a word
#Check if "Engineer" is present, but NOT at the end of a word
#Check if "ring" is present, but NOT at the end of a word

x = re.findall(r"\BABESEC", txt)

y = re.findall(r"Engineer\B", txt)

z = re.findall(r"ring\B", txt)

print(x)
print(y)
print(z)
```

Special Sequence for start

Special Sequence for end

```
['ABESEC']
['Engineer']
[]
```

Output

Example

Write a Python program to find all four characters long word in a string.

```
import re
text = 'ABES EC Campus:1, NH24, GHaziabad,UP'
print(re.findall(r"\b\w{4}\b", text))
```

```
['ABES', 'NH24']
```

Output

Example

Write a Python program to find all three, four, five and six characters long words in a string.

```
import re
text = 'ABES EC Campus:1, NH24, GHaziabad,UP'
print(re.findall(r"\b\w{3,6}\b", text))
```

```
['ABES', 'Campus', 'NH24']
```

Output

Example

Write a Python program to find all words which are at least 5 characters long in a string.

```
import re
text = 'ABES EC Campus:1, NH24, GHaziabad,UP'
print(re.findall(r"\b\w{5,}\b", text))
```

```
['Campus', 'GHaziabad']
```

Output

Session Plan - Day 5

5.2 Group(),

groups(),

sub(),

split(),

compile().

More Regex methods

Group () or Groups() –

We would use group(num)or groups() function of match object to get matched expression.

Syntax -

Cont..

Example 1: Write a program to print the username, org_name and domain from a emailID.

'()' parenthesis are used to define a specific group

```
import re
test='pythongroup@abes.ac.in'
match_object = re.match(r'(\w+)@(\w+)\.(\w+)', test)

print(match_object.group())
print(match_object.group(0))

print(match_object.group(1))

print(match_object.group(2))

print(match_object.group(3))

print(match_object.groups())
print(match_object.group(1, 2, 3))
```

for entire match

for the first subgroup

for the second subgroup

for the third subgroup

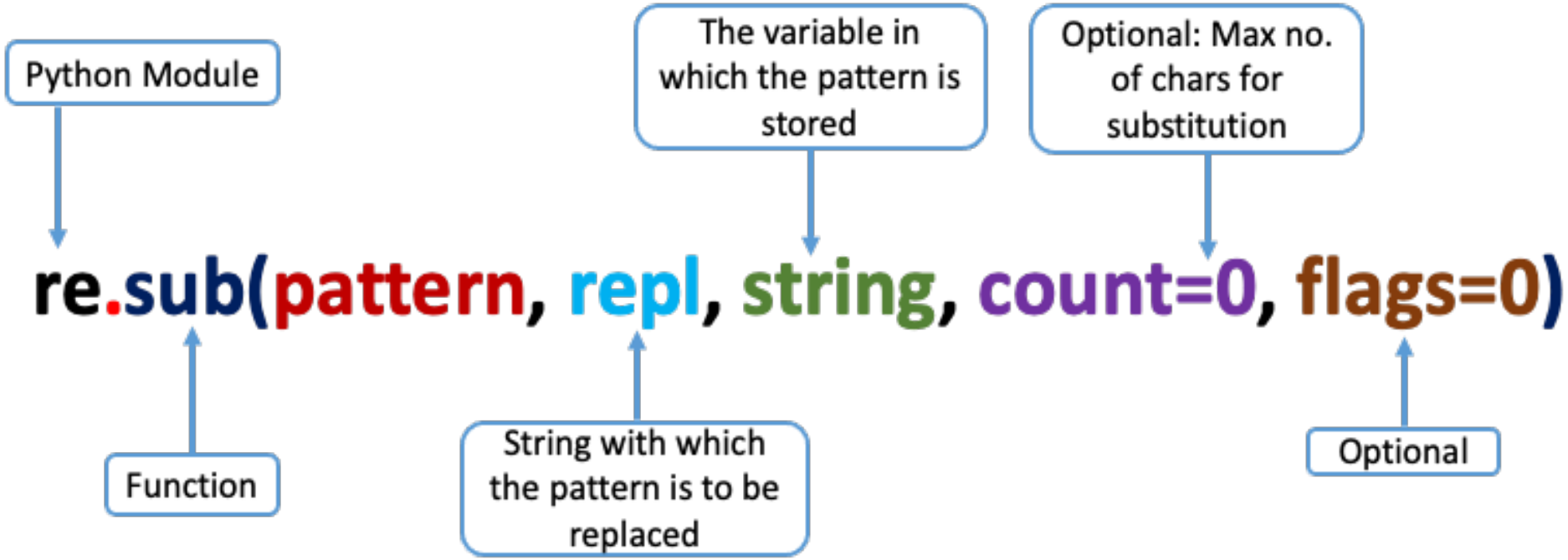
for a tuple of all matched subgroups

```
pythongroup@abes.ac
pythongroup@abes.ac
pythongroup
abes
ac
('pythongroup', 'abes', 'ac')
('pythongroup', 'abes', 'ac')
```

Output

re.sub() method

The re.sub() method replaces instances of a certain sub-string with another sub-string.



re.sub() method

Example - Write a program to demonstrate sub() method in regular expression.

```
import re

text = "ABESEC Campus 1, NH 24, GZB, UP"
output1 = re.sub("\s", "-", text)
print(output1)

output2 = re.sub("\s", ":", text, 2)
print(output2)
```

Replace space by -

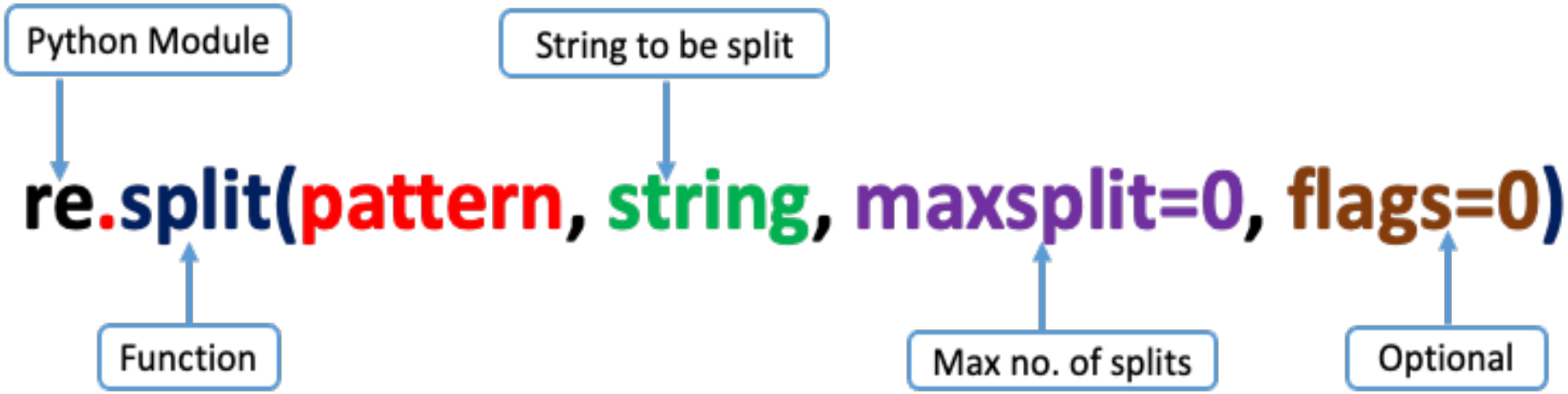
Replace the first 2 occurrences of space by :

```
ABESEC-Campus-1,-NH-24,-GZB,-UP
ABESEC:Campus:1, NH 24, GZB, UP
```

Output

re.split() method

The re.split() method split the string by the occurrences of the regex pattern, returning a list containing the resulting substrings.



re.split() method

Example 1: Write a program to demonstrate split() method in regular expression.

```
import re

text = "ABESEC: NH24, GZB, UP"

x = re.split("\s", text)

y = re.split("\s", text, maxsplit=2)
print(x)
print(y)
```

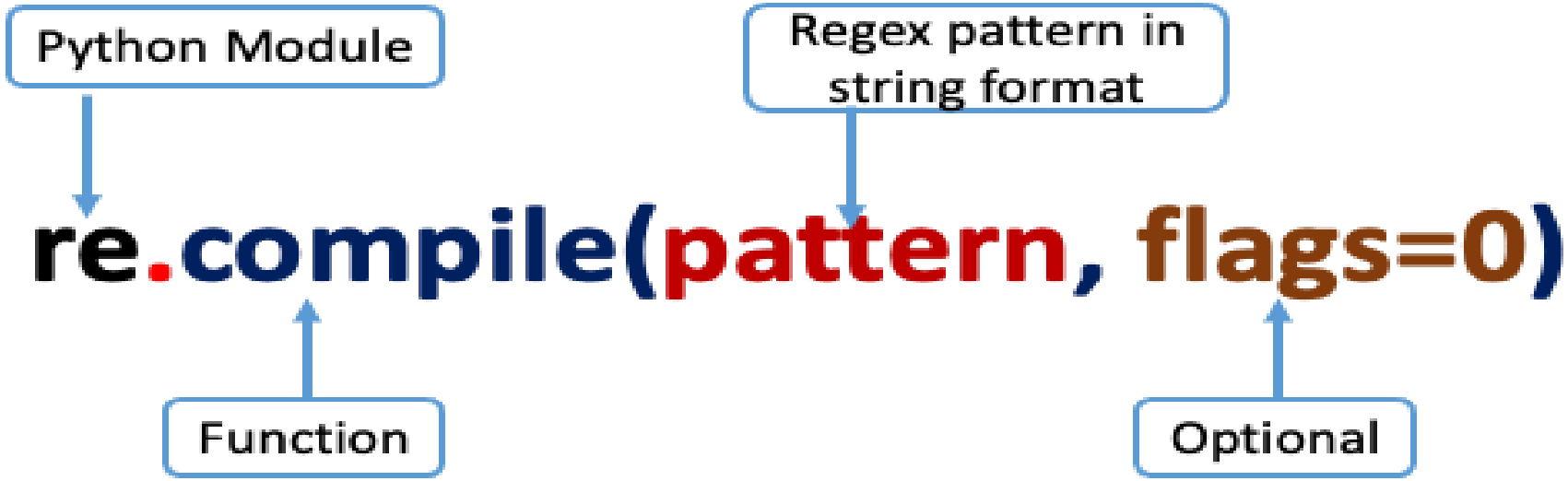
Most two splits

```
['ABESEC:', 'NH24,', 'GZB,', 'UP']
['ABESEC:', 'NH24,', 'GZB, UP']
```

Output

re.compile() method

The re.compile() method is to compile the regex pattern into pattern object (re.Pattern),which can be used for matching later.



re.compile() method

Example: Write a program to demonstrate compile() method in regular expression.

```
import re

# Target String
text = "ABESEC:Campus1, NH 24, GZB"
pattern = r"\d"

pattern_object = re.compile(pattern)
print(type(pattern_object))

result = pattern_object.findall(text)
print(result)
```

re.pattern object

Use Pattern object returned by the **compile() method** to match a regex pattern.

```
<class 're.Pattern'>
['1', '2', '4']
```

Output

Can you answer these questions?

Complete the given program to check if a string has at least 1 zero.

```
import re
txt = 'You Like Python'
s = re.findall("(\AI |Python\Z)",txt)
print(s)
```

a) ['Python']

b) [ ]

c) ['I Python']

d) Error

Can you answer these questions?

Complete the given program to check if a string has at least 1 zero.

```
import re
txt = 'I Love Python but I Like C'
s = re.findall("(\AI|Python\Z)",txt)
print(s)
```

a) ['Python']

b) ['I']

c) ['I Python']

d) Error

Can you answer these questions?

Which of the following creates a pattern object?

a) re.create(str)

b) re.regex(str)

c) re.compile(str)

d) re.assemble(str)

Can you answer these questions?

What will be the output of the following Python code?

```
re.split(',', 'I Love Python, but I Like C')
```

a) **['I Love Python', ' but I Like C']**

b) **['I', 'Love', 'Python', 'but', 'I', 'Like', 'C']**

c) **['I', 'Love', 'Python']**

d) **Error**

Summary

References

1. https://docs.python.org/3/tutorial/controlflow.html
2. Think Python: An Introduction to Software Design, Book by Allen B. Downey
3. Head First Python, 2nd Edition, by Paul Barry
4. Python Basics: A Practical Introduction to Python, by David Amos, Dan Bader, Joanna Jablonski, Fletcher Heisler
5. https://fresh2refresh.com/python-tutorial/python-jump-statements/
6. https://tutorialsclass.com/python-jump-statements/

Thank You