Python - Advance Level

Lesson 4 : Error Handling

What is Exception Handling

Let us begin!

What is an exception?

Definition of Exception

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions

What is exception handling?

Exception Handling

Process of responding to the occurrence, during computation, of exceptional conditions requiring special processing – often changing the normal flow of program execution

Common exceptions:

ImportError: an import fails

IndexError: a list is indexed with an out-of-range number

NameError: an unknown variable is used

SyntaxError: the code can not be parsed properly

TypeError: a function is called on a value of an inappropriate type.

ValueError: a function is called on a value of the correct type, but with an inappropriate value.

Process of Exception Handling

Important Terms

try	Keyword used to keep the code segment under check
except	Segment to handle the exception after catching it
else	Run this when no exceptions exist
finally	No matter what run this code if/if not for exception

Process of Exception Handling

If an error is encountered, a **try** block code execution is stopped and transferred down to the **except** block.

In addition to using an except block after the try block, you can also use the **finally** block.

The code in the finally block will be executed regardless of whether an exception occurs

Process of Exception Handling

Visually it looks like this!

```
try:
    Run this code
except:
    Execute this code when there is an exception
else:
    No exceptions? Run this code.
finally:
    Always run this code.
```

Exception Handling In Python

Coding In Python

Coding In Python

Python code for Exception Handling

```
>>> print( 0 / 0 ))
  File "<stdin>", line 1
    print( 0 / 0 ))
                  ^
SyntaxError: invalid syntax
```

Syntax Error

```
>>> print( 0 / 0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Exception

Exception Handling In Python

Try and Except Block

Try And Except Block

The try and except block in Python is used to catch and handle exceptions

```
try:
    Run this code
except:
    Execute this code when
    there is an exception
```

Try And Except Block

Another example where you open a file and use a built-in exception

```
try:
    with open('file.log') as file:
        read_data = file.read()
except:
    print('Could not open file.log')
```

If file.log does not exist, this block of code will output the following:

OUTPUT

```
Could not open file.log
```

Try And Except Block

In the Python docs, you can see that there are a lot of built-in exceptions that you can use

```
Exception FileNotFoundError

Raised when a file or directory is requested but doesn't exist.
Corresponds to errno ENOENT.
```

```
try:
    with open('file.log') as file:
        read_data = file.read()
except FileNotFoundError as fnf_error:
    print(fnf_error)
```

If file.log does not exist, this block of code will output the following:

OUTPUT

```
[Errno 2] No such file or directory: 'file.log'
```

Different ways to use error handling

```
try - Execution of code (block)
except - block lets you handle the Exception or Error
finally - it always execute
```

```
#way 1

try:
   data="aaa"
   print (data[1])
   print (12/0)
   print ("Hello")
   a=mohammed
except:
   print ("Error Occured")

print (“Technology Academy")
```

```
# way2

try:
   dat=121
   print (dat)
   print ("aa"[1.2])
   print (12/0)

except Exception as e:
   print (e)

print (“Technology Academy")
```

```
#way3

data="Aptech"
try:
   print (data[1.2])
   print (data[6])
   print (12/0)

except IndexError as e:
   print(e)
except TypeError as e:
   print (e)
```

Example

```
'''A module for demonstrating exceptions.'''

def convert(s):
    '''Convert to an integer.'''
    x = int(s)
    return x
```

```
'''A module for demonstrating exceptions.'''

def convert(s):
    '''Convert to an integer.'''
    try:
        x = int(s)
        print("Conversion succeeded! x =", x)
    except ValueError:
        print("Conversion failed!")
        x = -1
    return x
```

```
$ python3
Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more informati
on.
>>> from exceptional import convert
>>> convert("33")
33
>>> convert("hedgehog")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "./exceptional.py", line 5, in convert
    x = int(s)
ValueError: invalid literal for int() with base 10: 'hedgehog'
>>>
```

```
$ python3
Python 3.3.2 (v3.3.2:d047928ae3f6, May 13 2013, 13:52:24)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more informati
on.
>>> from exceptional import convert
>>> convert("34")
Conversion succeeded! x = 34
34
>>> convert("giraffe")
Conversion failed!
-1
>>>
```

Quiz

Quiz 1

Which number is not printed by this code?

- 3
- 2
- 4

```
try:
        print(1)
        print(20/0)
        print(2)
except ZeroDivisionError:
        print(3)
finally:
        print(4)
```

Quiz

2

Fill in the blanks to try to open and read from a file. Print an error message in case of an exception.

```
try:
      [    ] open("test.txt")as [  ]:
       print(f.read())
[    ]
       print("Error")
```

Quiz
3

What is an exception?

- A Variable
- An event that occurs due to incorrect code or input.
- function

Quiz
4

What is the output of this code?

- 1
- 1 3
- 1 2 3
- 3

```
try:
        print(1)
except :
        print(2)
finally:
        print(3)
```

Python - Advance Level

Lesson 5 : Regular Expression

Agendas

- Why we use Regular Expressions?
- What are Regular Expressions?
- Basic Regular Expressions operations
- E-mail verification using Regular Expressions
- Phone number verification using Regular Expressions
- Web scraping using Regular Expressions

Why We Use Regular Expressions?

Jul 1 03:27:12
syslog:
[m_java][
1/Jul/2013
03:27:12.818][j:
[SessionThread
<]^Iat
com/avc/abc/m
agr/service/find
.something(abc
/1235/locator/a
bc;Ljava/lang/S
tring;)Labc/abc/
abcd/abcd;(byt
ecode:7)

→ Date and Time

Why We Use Regular Expressions?

Sc # .com

sk@gmail.com

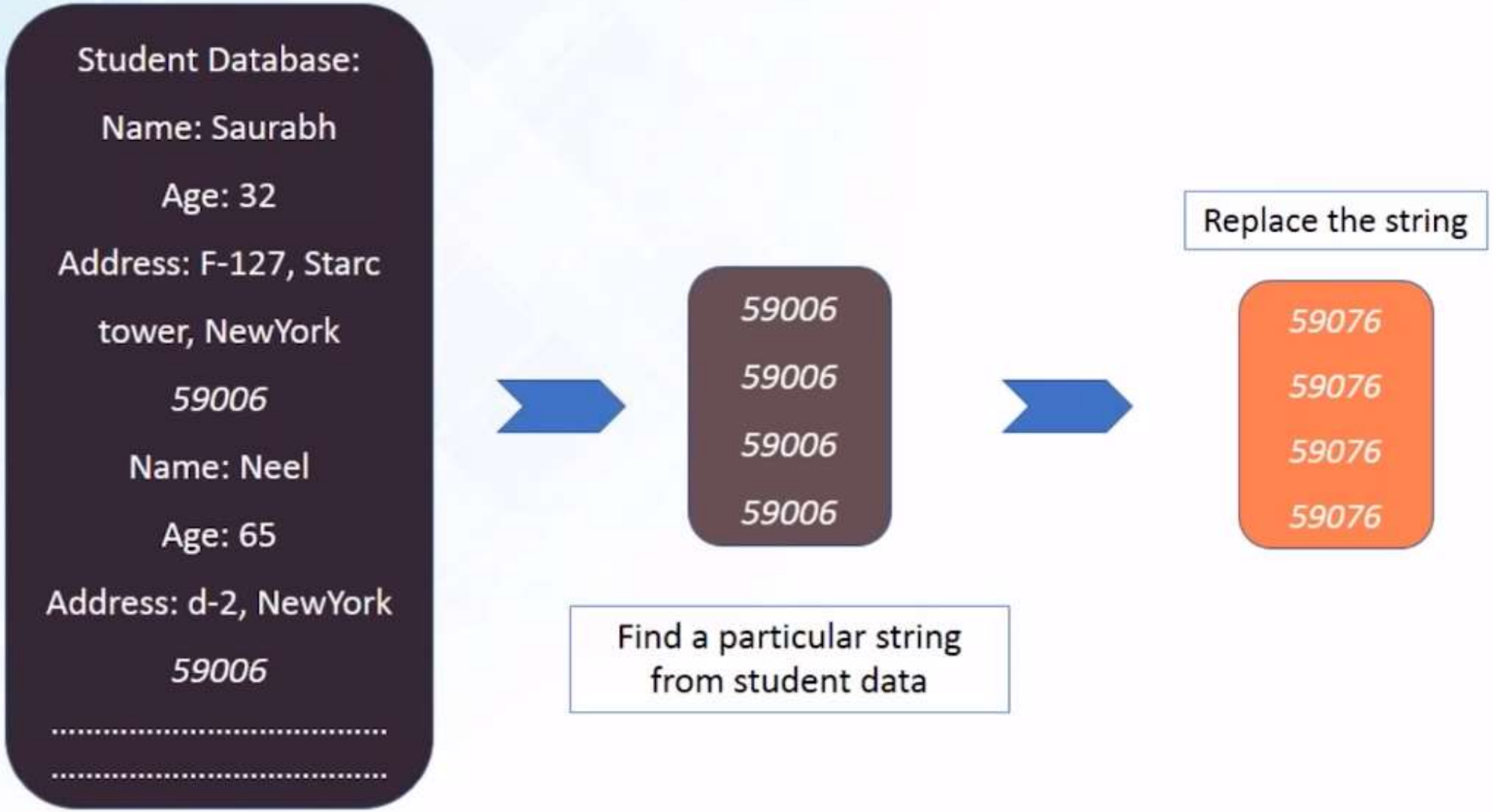
dk@gmail.com

sd@gmail.com

Cf @ cm

Why We Use Regular Expressions?

Why We Use Regular Expressions?



What are Regular Expressions?

A Regular Expression is a special text string for describing a search pattern.

Can you identify the pattern to get the Name and Age

```
NameAge = '''
Janice is 22 and Theon is 33
Gabriel is 44 and Joey is 21
'''
```

{'Janice': '22', 'Theon': '33', 'Gabriel': '44', 'Joey': '21'}

```
ages = re.findall(r'\d{1,3}', NameAge)
names = re.findall(r'[A-Z][a-z]*', NameAge)
```

First letter of all the Names is an uppercase letter and Age is represented by numbers

What are Regular Expressions?

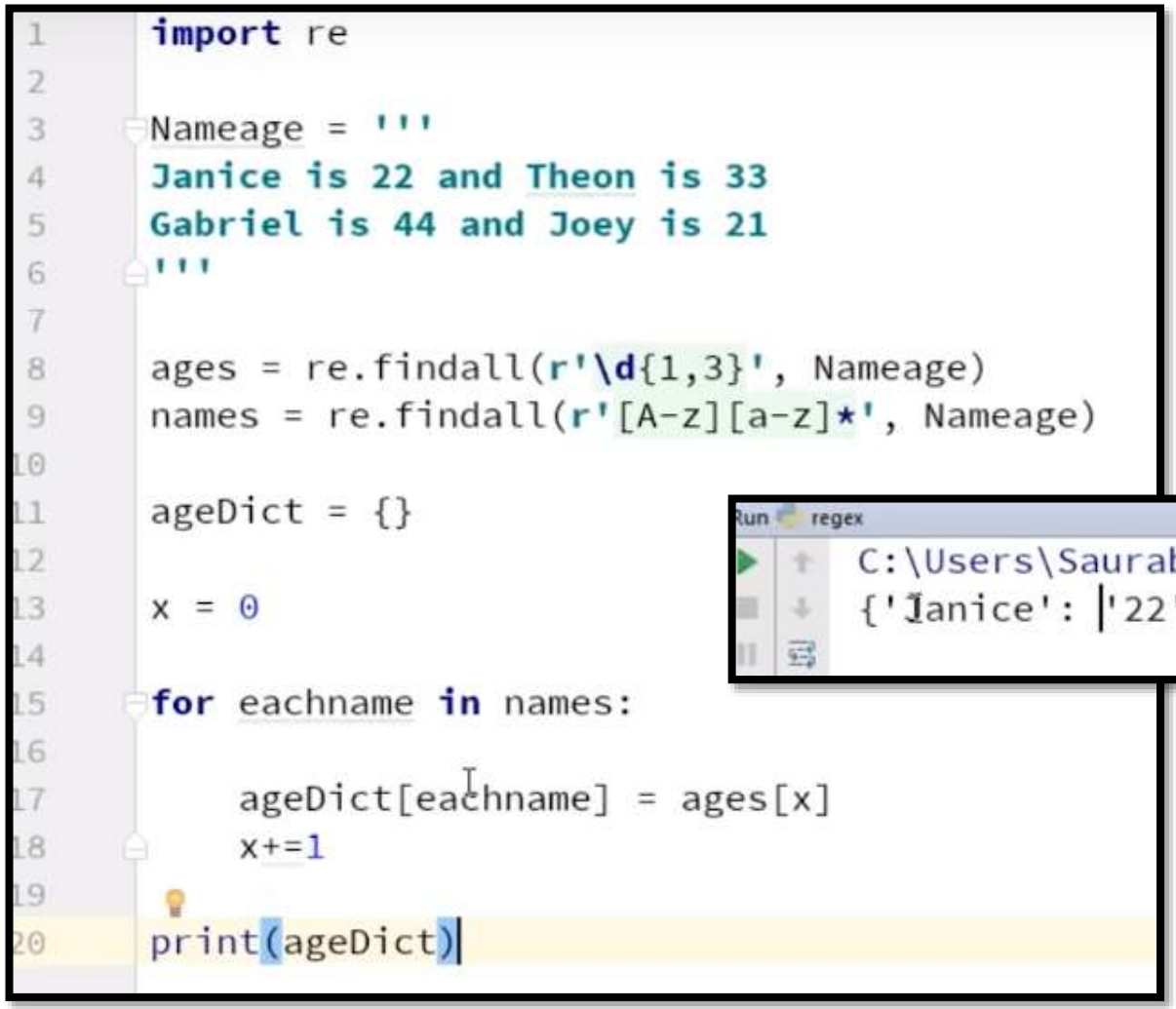
Regular expressions are a powerful tool for various kinds of string manipulation.

They are useful for two main tasks :

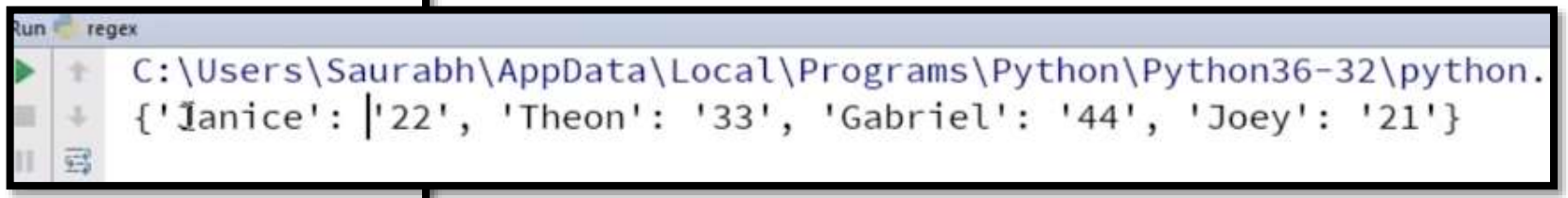
- verifying that string match a pattern (for instance, that a string has the format of an email address)
- Performing substitutions in a string (such as changing all American spellings to British ones.).

Regular expressions in Python can be accessed using the **re** module, with is part of the standard library.

Regular Expression



```python
import re

Nameage = '''
Janice is 22 and Theon is 33
Gabriel is 44 and Joey is 21
'''

ages = re.findall(r'\d{1,3}', Nameage)
names = re.findall(r'[A-z][a-z]*', Nameage)

ageDict = {}

x = 0

for eachname in names:

    ageDict[eachname] = ages[x]
    x+=1

print(ageDict)
```



```
Run  regex
C:\Users\Saurabh\AppData\Local\Programs\Python\Python36-32\python.
{'Janice': '22', 'Theon': '33', 'Gabriel': '44', 'Joey': '21'}
```

Operations that you can perform with Regular Expressions

ReEx Operations (match)

After you've defined a regular expression, the **re.match** function can be used to determine whether it matches at the **beginning** of a string.

If it does, **match** returns an object representing the match, if not, it returns **None**.

Example:

```
import re

pattern = "spam"

if re.match(pattern, "spamspamspam"):
        print("Match")
else:
        print("No match")
```

Result:

```
>>>
Match
>>>
```

ReEx Operations (search,findall)

Other functions to match patterns are **re.search** and **re.findall**.

The function **re.search** finds a match of a pattern anywhere in the string.
The function **re.findall** returns a list of all substrings that match a pattern.

```
import re

pattern = "spam"

if re.match(pattern, "eggspamsausagespam"):
        print("Match")
else:
        print("No match")

if re.search(pattern, "eggspamsausagespam"):
        print("Match")
else:
        print("No match")

print(re.findall(pattern, "eggspamsausagespam"))
```

```
>>>
No match
Match
['spam', 'spam']
>>>
```

ReEx Operations (Replace)

One of the most important **re** methods that use regular expressions is **sub**.
Syntax:

re.**sub**(pattern, repl, string, count=0)

This method replaces all occurrences of the **pattern** in string with **repl**, substituting all occurrences, unless **count** provided. This method returns the modified string.

```
import re

str = "My name is David. Hi David."
pattern = "David"
newstr = re.sub(pattern, "Amy", str)
print(newstr)
```

```
>>>
My name is Amy. Hi Amy.
>>>
```

Metacharacters

Metacharacters are what make regular expressions more powerful than normal string methods. They allow you to create regular expressions to represent concepts like "one or more repetitions of a vowel".

The first metacharacter we will look at is . (dot).

This matches **any character**, other than a new line.

```
import re

pattern = r"gr.y"

if re.match(pattern, "grey"):
        print("Match 1")

if re.match(pattern, "gray"):
        print("Match 2")

if re.match(pattern, "blue"):
        print("Match 3")
```

```
>>>
Match 1
Match 2
>>>
```

Metacharacters

The next two metacharacters are **^** and **$**.
These match the **start** and **end** of a string, respectively.

```
import re

pattern = r"^gr.y$"

if re.match(pattern, "grey"):
        print("Match 1")

if re.match(pattern, "gray"):
        print("Match 2")

if re.match(pattern, "stingray"):
        print("Match 3")
```

```
>>>
Match 1
Match 2
>>>
```

Metacharacters

Character Classes

Character classes provide a way to match only one of a specific set of characters.
A character class is created by putting the characters it matches inside **square brackets**.

```
mport re

pattern = "[aeiou]"

if re.search(pattern, "grey"):
        print("Match 1")

if re.search(pattern, "qwertyuiop"):
        print("Match 2")

if re.search(pattern, "rhythm myths"):
        print("Match 3")
```

```
>>>
Match 1
Match 2
>>>
```

Metacharacters

Character classes can also match ranges of characters.

Some examples:

-The class **[a-z]** matches any lowercase alphabetic character.
-The class **[G-P]** matches any uppercase character from G to P.
-The class **[0-9]** matches any digit.
-Multiple ranges can be included in one class. For example, **[A-Za-z]** matches a letter of any case.

```
import re

pattern = r"[A-Z][A-Z][0-9]"

if re.search(pattern, "LS8"):
        print("Match 1")

if re.search(pattern, "E3"):
        print("Match 2")

if re.search(pattern, "1ab"):
        print("Match 3")
```

```
>>>
Match 1
>>>
```

Metacharacters

Place a **^** at the start of a character class to **invert** it.
This causes it to match any character other than the ones included.
Other metacharacters such as **$** and **.**, have no meaning within character classes.
The metacharacter **^** has no meaning unless it is the first character in a class.

```
import re

pattern = r"[^A-Z]"

if re.search(pattern, "this is all quiet"):
print("Match 1")

if re.search(pattern, "AbCdEfG123"):
print("Match 2")

if re.search(pattern, "THISISALLSHOUTING"):
print("Match 3")
```

```
>>>
Match 1
Match 2
>>>
```

Metacharacters

Some more metacharacters are *****, **+**, **?**, **{** and **}**.
These specify numbers of repetitions.
The metacharacter ***** means "zero or more repetitions of the previous thing". It tries to match as many repetitions as possible. The "previous thing" can be a single character, a class, or a group of characters in **parentheses**.

```
import re

pattern = r"egg(spam)*"

if re.match(pattern, "egg"):
print("Match 1")

if re.match(pattern, "eggspamspamegg"):
print("Match 2")

if re.match(pattern, "spam"):
print("Match 3")
```

```
>>>
Match 1
Match 2
>>>
```

Metacharacters

The metacharacter **+** is very similar to *****, except it means "**one** or more repetitions", as opposed to "zero or more repetitions".

```
import re

pattern = r"g+"

if re.match(pattern, "g"):
print("Match 1")

if re.match(pattern, "ggggggggggggg"):
print("Match 2")

if re.match(pattern, "abc"):
print("Match 3")
```

```
>>>
Match 1
Match 2
>>>
```

Metacharacters

The metacharacter **?** means "zero or one repetitions".

```
import re

pattern = r"ice(-)?cream"

if re.match(pattern, "ice-cream"):
print("Match 1")

if re.match(pattern, "icecream"):
print("Match 2")

if re.match(pattern, "sausages"):
print("Match 3")

if re.match(pattern, "ice--ice"):
print("Match 4")
```

```
>>>
Match 1
Match 2
>>>
```

Metacharacters

Curly braces can be used to represent the number of repetitions between two numbers.
The regex **{x,y}** means "between x and y repetitions of something".
Hence **{0,1}** is the same thing as **?**.
If the first number is missing, it is taken to be zero. If the second number is missing, it is taken to be infinity.

```
import re

pattern = r"9{1,3}$"

if re.match(pattern, "9"):
print("Match 1")

if re.match(pattern, "999"):
print("Match 2")

if re.match(pattern, "9999"):
print("Match 3")
```

```
>>>
Match 1
Match 2
>>>
```

Metacharacters

https://regexone.com/lesson/introduction_abcs

Another important metacharacter is **|**.
This means "or", so **red|blue** matches either "red" or "blue".

```
import re

pattern = r"gr(a|e)y"

match = re.match(pattern, "gray")
if match:
print ("Match 1")

match = re.match(pattern, "grey")
if match:
print ("Match 2")

match = re.match(pattern, "griy")
if match:
print ("Match 3")
```

```
>>>
Match 1
Match 2
>>>
```

https://regexone.com/lesson/introduction_abcs

special sequences

special sequences are **\d**, **\s**, and **\w**.

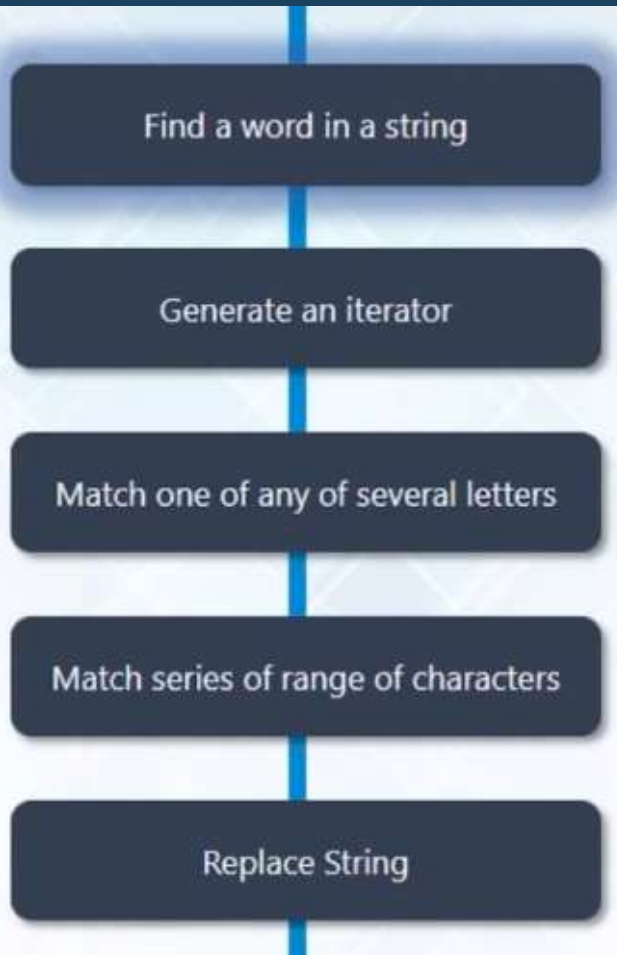
These match **digits**, **whitespace,** and **word characters** respectively.

In ASCII mode they are equivalent to [0-9], [\t\n\r\f\v], and [a-zA-Z0-9_].

In Unicode mode they match certain other characters, as well. For instance, **\w** matches letters with accents.

Versions of these special sequences with upper case letters - **\D**, **\S**, and **\W** - mean the opposite to the lower-case versions. For instance, **\D** matches anything that isn't a digit.

Regular Expression Operations



Find a word in a string

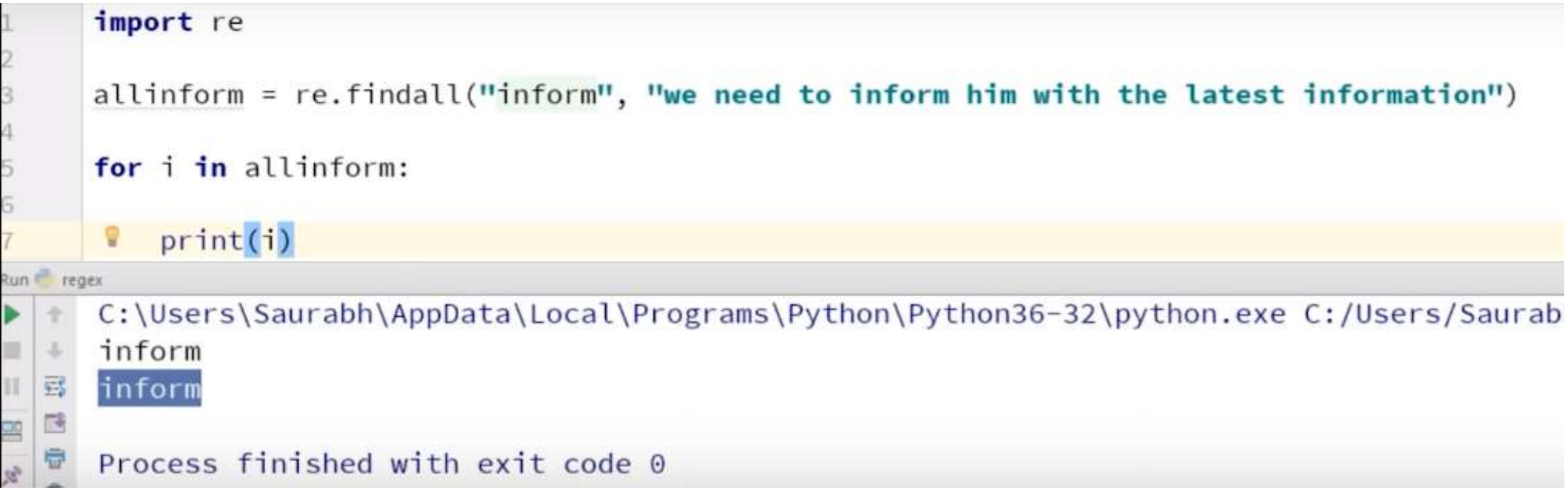
```
import re

allinform = re.findall("inform", " we need to inform him with the latest
information ")

for i in allinform:
    print(i)
```

Code Example (re.search)

```
import re

if re.search("inform", "we need to inform him with the latest information"):

    print("There is inform")
```

```
Run  regex
C:\Users\Saurabh\AppData\Local\Programs\Pytho
There is inform

Process finished with exit code 0
```

Code Example (re.findall)



```
import re

allinform = re.findall("inform", "we need to inform him with the latest information")

for i in allinform:

    print(i)
```

Run regex

```
C:\Users\Saurabh\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/Saurab
inform
inform

Process finished with exit code 0
```

Regular Expression Operations

- Find a word in a string
- Generate an iterator
- Match one of any of several letters
- Match series of range of characters
- Replace String

Get the starting and ending index of a particular string

```
Str = "we need to inform him with the latest information"

for i in re.finditer("inform.", Str):

    locTuple = i.span()

    print(locTuple)
```

Code Example (re.finditer)

```python
import re

str = "we need to inform him with the latest information"

for i in re.finditer("inform", str):

    loctup = i.span()

    print(loctup)
```

```
regex
C:\Users\Saurabh\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/Sau
(11, 17)
(38, 44)
```

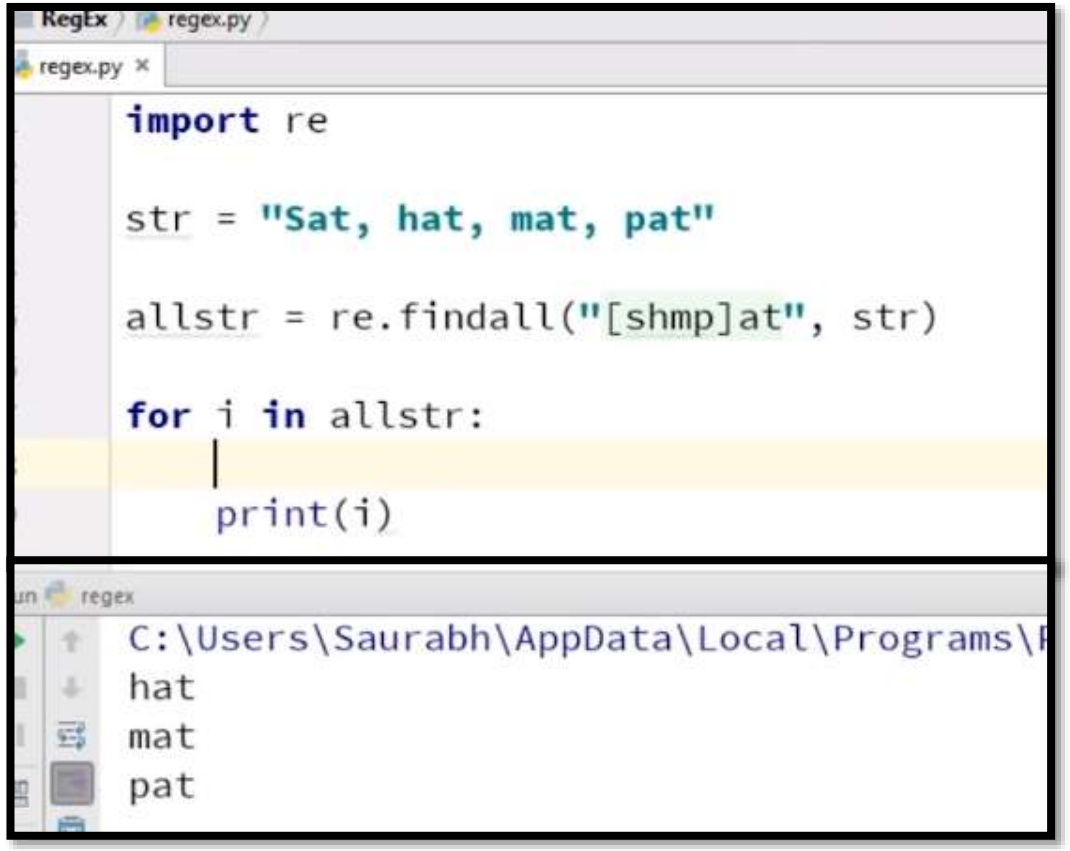
Regular Expression Operations

- Find a word in a string
- Generate an iterator
- Match one of any of several letters
- Match series of range of characters
- Replace String

Match words with a particular pattern

```
Str = "Sat, hat, mat, pat"

allStr = re.findall("[shmp]at", Str)

for i in allStr:
    print(i)
```

Regular Expression (re.findall)



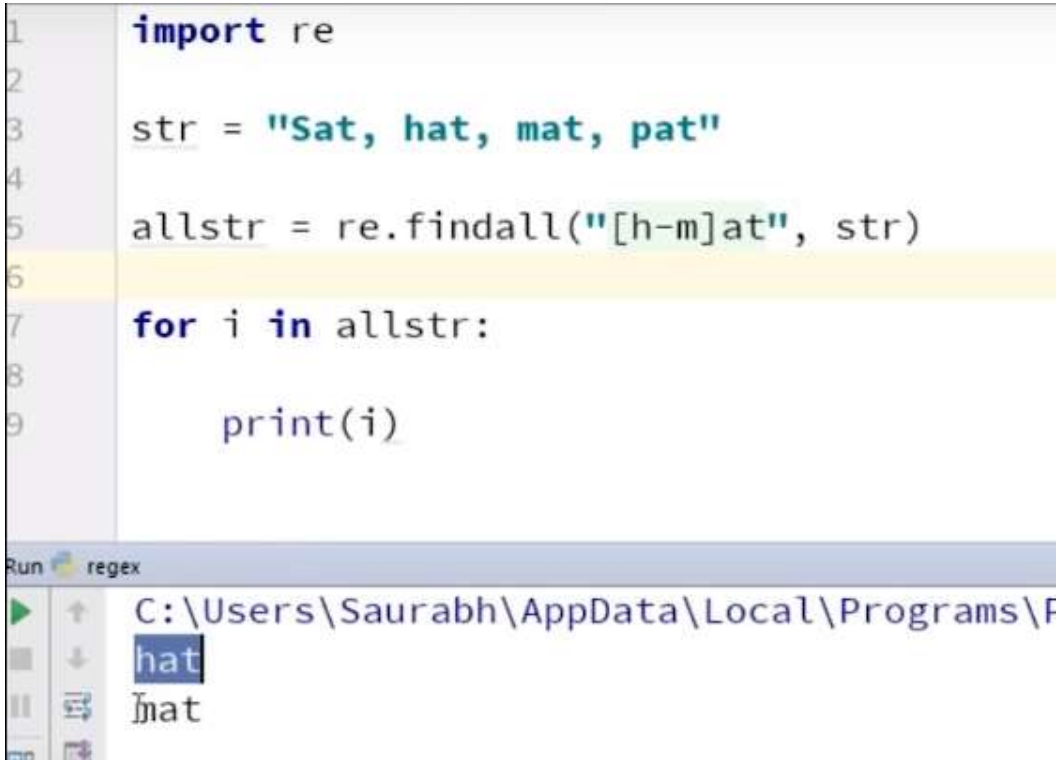
Regular Expression Operations

- Find a word in a string
- Generate an iterator
- Match one of any of several letters
- Match series of range of characters
- Replace String

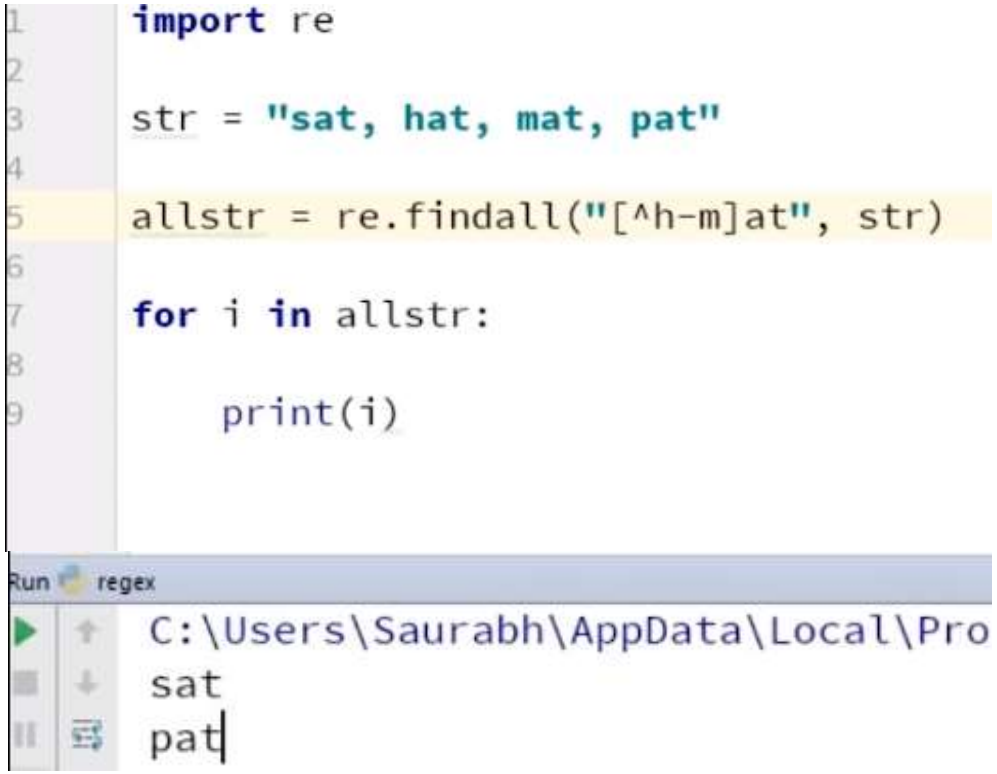
Match series of range of characters

```
import re

Str = "Sat, hat, mat, pat"

someStr = re.findall("[h-m]at", Str)

for i in someStr:
    print(i)
```

Regular Expression



Regular Expression



Regular Expression Operations

Replace a string

```
import re

str = "My name is David. Hi David.“

pattern = "David“

newstr = re.sub(pattern, "Amy", str)

print(newstr)
```

Regular Expressions Applications

Phone Number Vertification

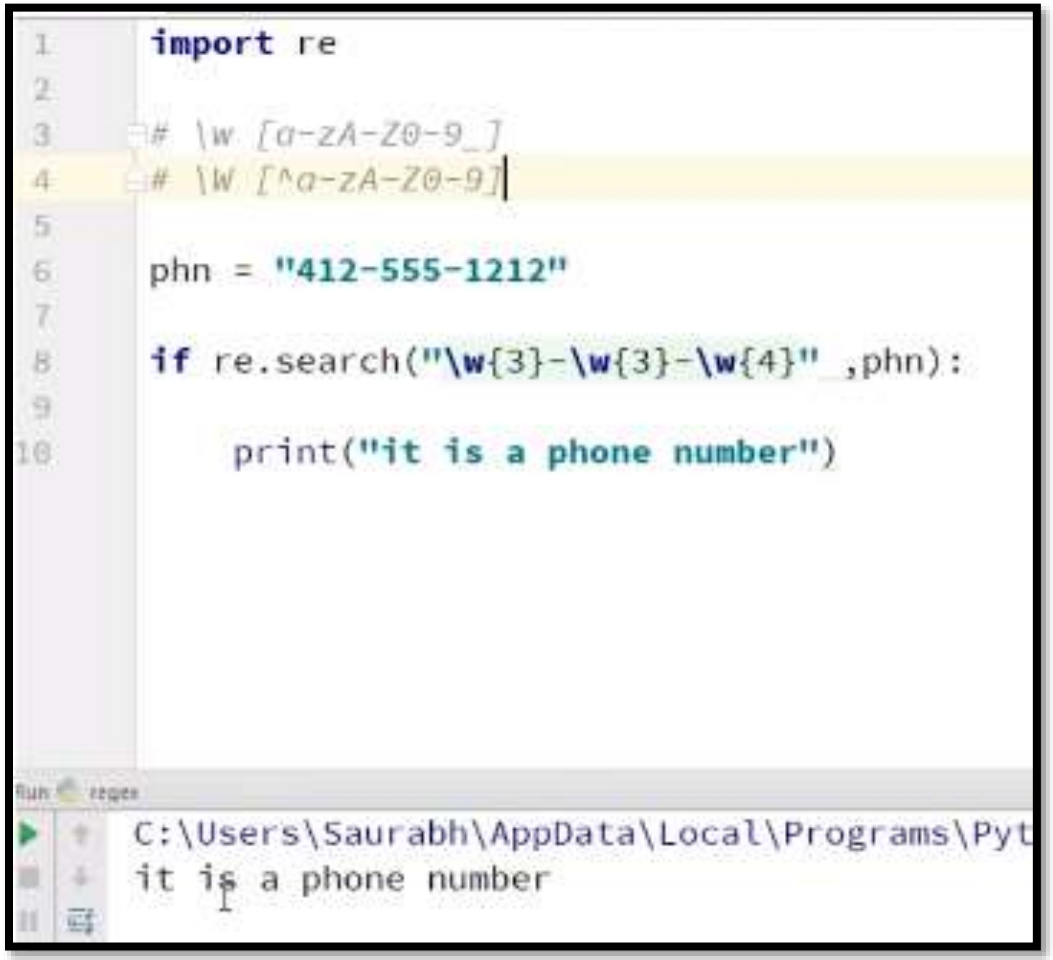
Here we are supposed to verify the phone numbers

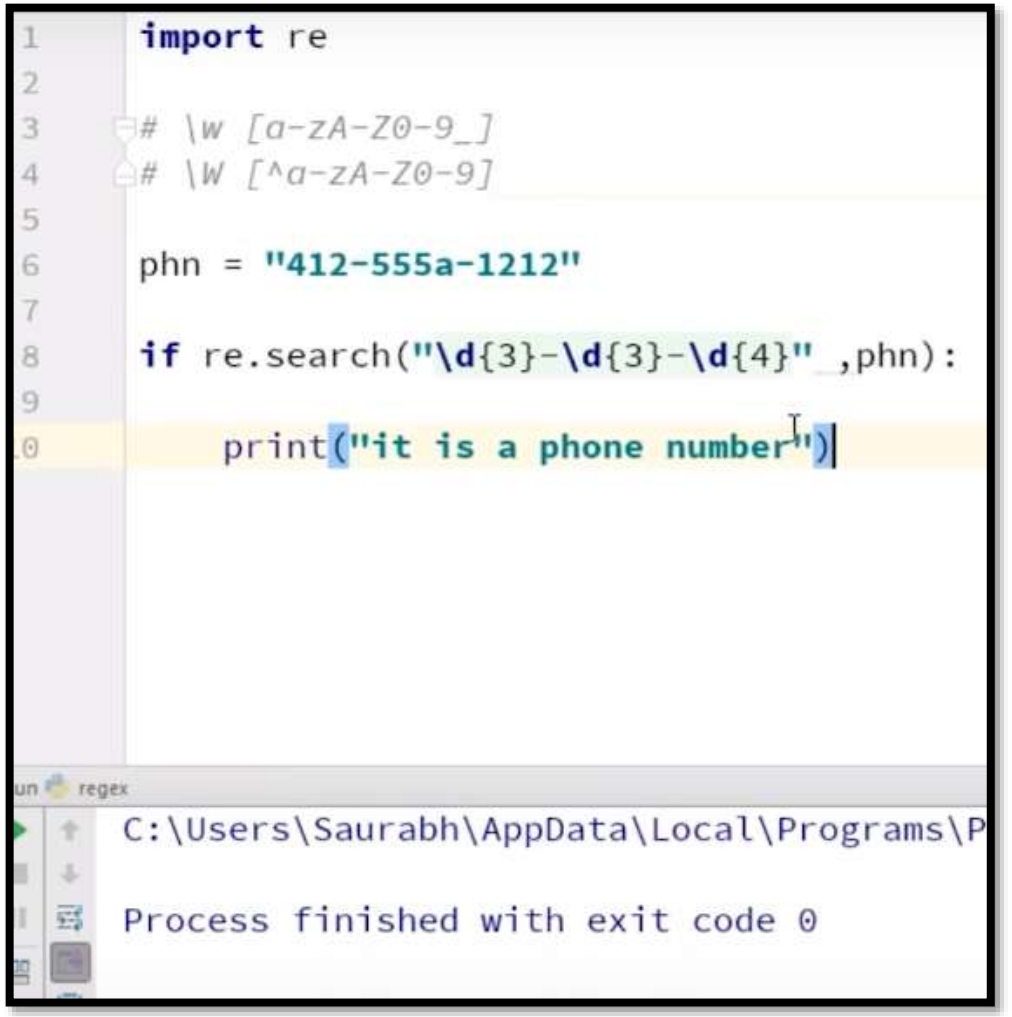
Problem Statement:
To verify the phone numbers

Phone Number
444-122-1234
123-122-78999
111-123-23
67-7890-2019

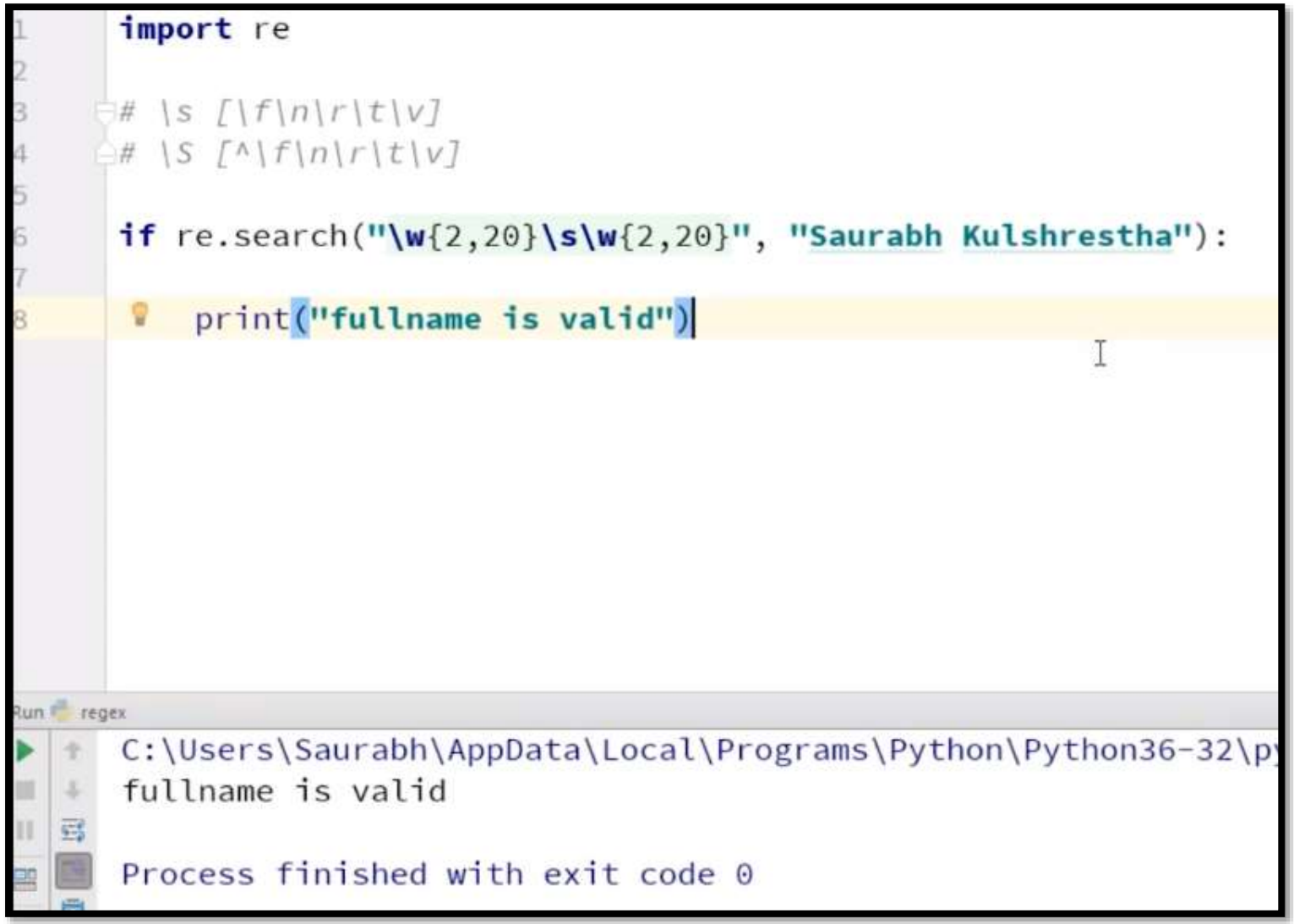
All phone numbers should have:
- 3 starting digits and '-' sign
- 3 middle digits and '-'sign
- 4 digits in the end

Solution


import re
# \w [a-zA-Z0-9_]
# \W [^a-zA-Z0-9]
phn = "412-555-1212"
if re.search("\w{3}-\w{3}-\w{4}" ,phn):
print("it is a phone number")
C:\Users\Saurabh\AppData\Local\Programs\Pyt
it is a phone number


import re
# \w [a-zA-Z0-9_]
# \W [^a-zA-Z0-9]
phn = "412-555a-1212"
if re.search("\d{3}-\d{3}-\d{4}" ,phn):
print("it is a phone number")
C:\Users\Saurabh\AppData\Local\Programs\P
Process finished with exit code 0

Another example



Quiz

Quiz 1

Which module in Python supports regular expressions?

a) re
b) regex
c) pyregex
d) none of the mentioned

Quiz
2

What does the function re.match do?

a) matches a pattern at the start of the string
b) matches a pattern at any position in the string
c) such a function does not exist
d) none of the mentioned

E-main Verification

Recall the second problem where we were supposed to verify the E-mail addresses

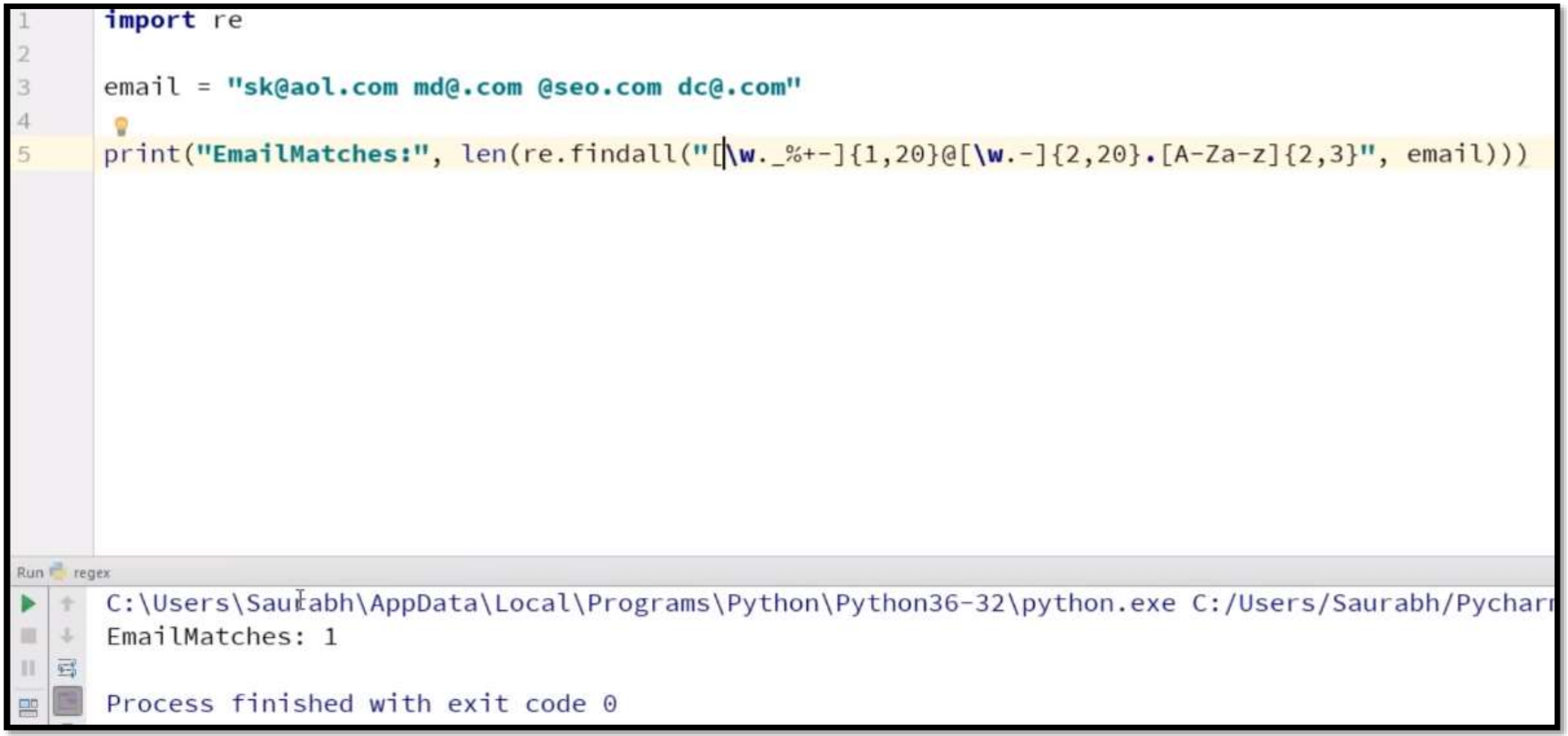
Problem Statement:
To verify the E-main addresses

E-mail:
Saurabh@gmail.com
Reyshma @ com
Kv .com
123 @.com

E-mail address should include:

- 1 to 20 lowercase and uppercase letters, numbers, plus ._%+-
- An @ symbol
- 2 to 20 lowercase and uppercase letters, numbers, plus .-
- A period
- 2 to 3 lowercase and uppercase letters

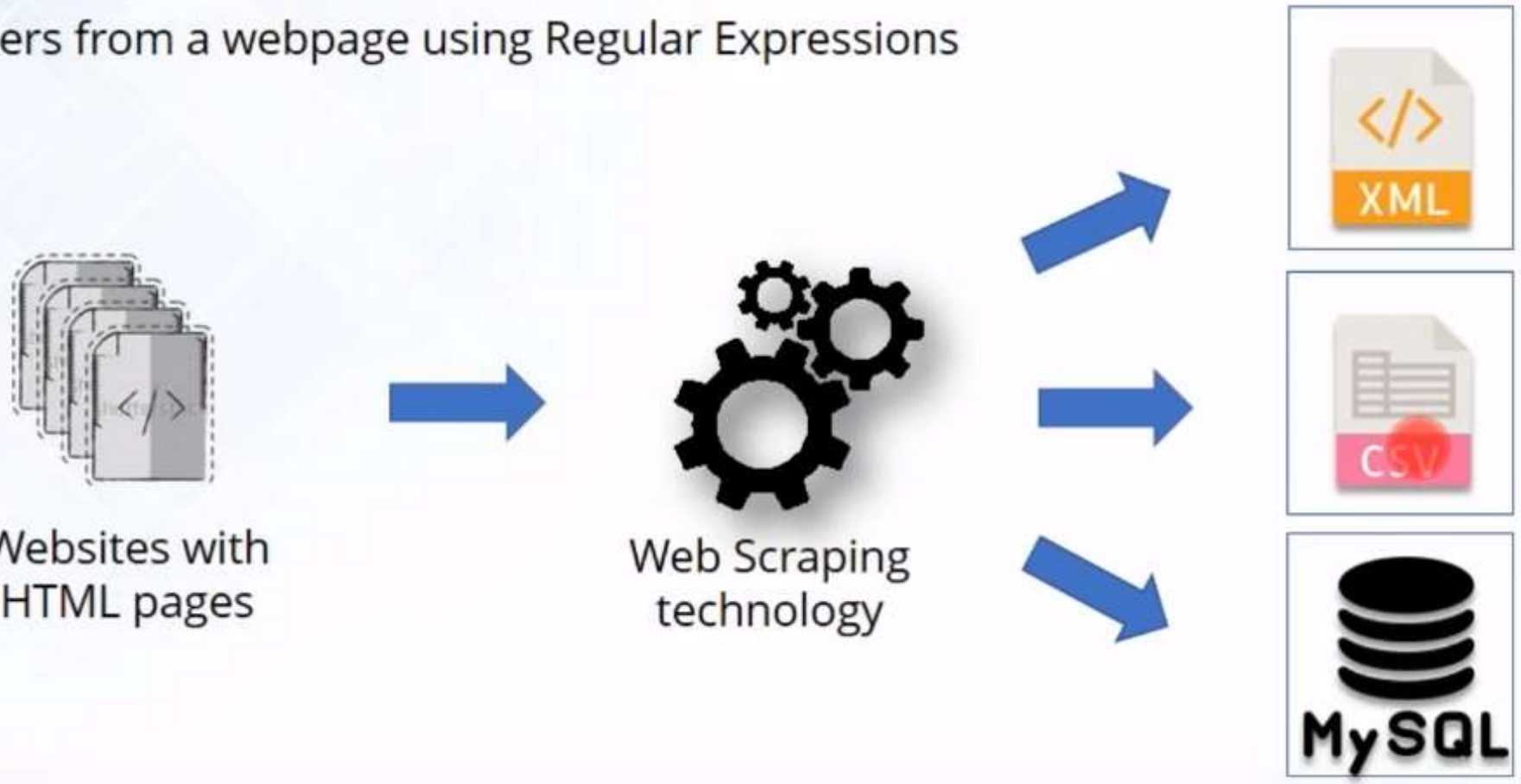
Solution


1 import re
2
3 email = "sk@aol.com md@.com @seo.com dc@.com"
4
5 print("EmailMatches:", len(re.findall("[\w._%+-]{1,20}@[\w.-]{2,20}.[A-Za-z]{2,3}", email)))
Run regex
C:\Users\Saurabh\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/Saurabh/Pycharm
EmailMatches: 1
Process finished with exit code 0

Web Scraping

Problem Statement:

Scrap all the phone numbers from a webpage using Regular Expressions



Solution

```python
import urllib.request
from re import findall

url = "http://www.summet.com/dmsi/html/codesamples/addresses.html"

response = urllib.request.urlopen(url)

html = response.read()

htmlStr = html.decode()

pdata = findall("\(\d{3}\) \d{3}-\d{4}", htmlStr)

for item in pdata:
   print(item)

```