

Python Training



Day -1

Introduction



General Instructions

- For Hands-on, follow the instructions provided in [Bitbucket Training](#).
- An example for a quick reference,
 - Repo: { Your college name }
 - Branch: jay-joshi-python-day-3
 - Folder structure: Jay Joshi / Python / Day-1 / hands-on-1-calculator.py
 - Commit message: “Jay Joshi: Python: Day-3: Rest API”
 - Pull Request named “Jay Joshi: Python: Day3”
 - Branch: jay-joshi-python-day-1 → Master
 - Description:
 - Hands-on requirement
 - List of scenarios tested (if possible, attach screenshots)
 - Reviewer: Your respective mentors.
 - Check “Delete after the pull request is merged”

Introduction to Python

- **What is a scripting language :** The scripting language is basically a language where instructions are written for a run time environment. They do not require the compilation step and are rather interpreted. [\[Ref\]](#)
- **What is interpreted language :** An interpreted programming language is a language designed to execute source code directly and without the need to compile a program into machine-language instructions. It is executed line-by-line at runtime, instead of compiling the whole source code before the execution [\[Ref\]](#)
- **What is high level language :** The high-level language is a programming language that allows a programmer to write the programs which are independent of a particular type of computer. The high-level languages are considered as high-level because they are closer to human languages than machine-level languages (easier to read write and maintain).
- **Python :** Python is a widely used general-purpose, high-level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed to improve the code readability, and its syntax allows programmers to express concepts in fewer lines of code.
 - **Application & Advantages :** [\[Ref\]](#)
 - **Python Installation -** [\[Ref\]](#) [\[Download Link\]](#)

Introduction to Python

- Python : [Basic Syntax](#), [Comments](#)
- [Strings](#), [Numbers](#), [Conversions](#).
- [Python Operators](#) (7 types)
- [Loops in Python](#)
- [Conditional Statements in Python](#), [Switch Case](#)
- [Python Functions](#)
- Inputs from Console: [[Run time Input](#)] [[System Arguments](#)]
- Best practices: naming conventions, constants, consistency in design : [[Ref](#)]
- Zen of python:
 - <https://rahul-saini.medium.com/the-origins-zen-of-python-b31a7296dbe0#0f5b>

References.

- <https://thepythonguru.com/is-python-a-programming-language-or-scripting-language/>
- https://www.tutorialspoint.com/python/python_classes_objects.htm
- https://www.w3schools.com/python/python_booleans.asp
- <https://wiki.python.org/moin/BeginnersGuide/Download>
- <https://www.datacamp.com/community/tutorials/functions-python-tutorial>
- <https://medium.com/better-programming/contemplating-the-zen-of-python-186722b833e5>

Day - 1

Data Types



Data Types

- Text types and Numeric Types (str, int, float)
 - How to specify the float numbers in scientific notation?
 - Different str operations - [Split](#), slice, [Strip](#)
- Sequence types ([list](#), [tuple](#))
 - list vs tuple
 - [Mutability](#)
 - When to use which type?
- [Python Sets](#)
- Mapping type ([dict](#))
- [Python Map Function](#)
- [Iterators](#) & [Generators](#) : [[Ref](#)]

Summary:

- List is a collection which is ordered and changeable. Allows duplicate members.
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
- Set is a collection which is unordered, unchangeable, and unindexed. No duplicate members.
- Dictionary is a collection which is ordered and changeable. No duplicate members.

References

1. Data Types in Python - https://www.w3schools.com/python/python_datatypes.asp (This contains detailed operations all the data types and the methods that can be used with each data types.)
2. Generators in Python - <https://www.geeksforgeeks.org/generators-in-python/>
3. Why generator? - <https://www.programiz.com/python-programming/generator>
4. How to define an iterator: <https://www.programiz.com/python-programming/iterator>
5. How to define a generator: <https://www.programiz.com/python-programming/generator>

Additional References:

1. Data types - <https://realpython.com/python-data-types/>
2. List vs tuple - <https://www.geeksforgeeks.org/python-difference-between-list-and-tuple/>

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day-1 Hands-on Python Basic



Introduction - Handson

1. Write python script to take no of arguments as input from the user. Then read the arguments from the standard input. Print read arguments on output.
2. Write python script to take one integer argument and then print as follows:
 - If Value >0 and Value < 10 — Small
 - If Value > 10 and Value <100 — Medium
 - If Value <1000 — Large
 - If Value > 1000 — Invalid
1. Write a function to find larger of three numbers.
 - Functions for each possible way we can find larger of three numbers.
2. Write a program to take two numbers as input parameter and then ask for the arithmetic parameter to be performed.

```
>>> "Enter Two numbers"
10 45
>>>"Operations to perform "
+
>>> 55
```
3. Write a program to take two integers as input. Print those two integers as output and then call a function to swap those two integers.
 - Write function for each possible way to swap two integers

Introduction - Handson

6. Write a python function which counts the frequency of given character in a given string. Inputs - A String A Character whose frequency needs to be determined
7. Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return "Empty String"
8. Write a Python program to add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged.
9. Write a Python function to remove the characters which have odd index values of a given string.
10. Write a Python function to insert a string in the middle of a string. For odd length of string, remove the middle character and replace with given string.

Doubt during Hands-on

- Purpose of Google chat training group.
- Free to ask relevant questions of training
- All member are highly encourage to reply the questions/doubt
- Share the Articles/Training videos

Introduction - Handson

11. Write a program a function for ATM machine which takes amount as input and output should be number of notes of each denomination. The ATM has notes in following denomination : 2000, 500, 100. Note that the ATM machine rarely gives all notes of a single amount. If you enter 4000, it will give 1 2000rs, 3 500rs and 5 100rs notes for even distribution.
11. Read a sentence from the standard input. Find out how many times each word appear in given string.

```
>>> Input :
```

```
"This is a Python learning"
```

```
>>> Output:
```

```
>>> This 1
```

```
>>> Is 1
```

```
>>> a 1
```

```
>>> Python 1
```

```
>>> Learning 1
```

Introduction - Handson

13. Paresh owns a company that moves containers between two islands. He has N trips booked, and each trip has P containers. Paresh has M boats for transporting containers, and each boat's maximum capacity is C containers. Given the number of containers going on each trip, determine whether or not Paresh can perform all trips using no more than boats per individual trip. If this is possible, print Yes; otherwise, print No.

Input Format

The first line contains three space-separated integers describing the respective values of N (number of trips), C (boat capacity), and M (total number of boats).

The second line contains space-separated integers describing the value for container for each trip.

Constraints

* $1 \leq m, c, p \leq 100$

Output Format

Print Yes if Paresh can perform booked trips using no more than boats per trip; otherwise, print No.

Sample Input 0 5 2 2 1 2 1 4 3	Sample Input 0 5 1 2 1 2 1 4 3
Sample Output 0 Yes	Sample Output 0 No

Day-1 Hands-on Data Types



Data Types - Handson

1) Saurabh needs to withdraw X Rs. from an ATM. The transaction will succeed only if X is an odd number, and Saurabh's account balance has enough cash to perform the withdrawal transaction (including bank charges). For each successful withdrawal the bank charges 10.50 Rs. Calculate Saurabh's account balance after an attempted transaction.

Input:

- Saurabh's initial account balance
- Withdrawal amount

Output

- Amount present in Saurabh's account after withdrawal.
- Error message, if the withdrawal did not match transaction criteria.

2) Write a program to take size of the list as input. Then read the integer values and store these details into list.

Output:

- The list entered by the user.

3) Write a program that takes Student's name and their marks in 3 subjects as input. Print each student's total marks as output.

4) Write a program that takes the input from the user (i.e., N). Create the generator function that takes this input as an argument and returns numbers from 1 to N.

Output:

- Using the generator function, print the numbers from 1 to N.

Data Types - Handson

5) Iterate on the dictionary made in Program #3 and find the percentage of each student, store it and print it in the console.
Use iterables objects to store the values.

6) Create a program to take student information as input. Student will have First Name, Last Name, Roll No. Write a function to sort the list based on given input parameter.

Input Parameters can be: 'By First Name' or 'Last Name' or 'Roll No'.

7) Write a Python program to input a list of non-empty tuples, sort it in increasing order by the last element in each tuple.

Sample List : [(2, 5), (1, 2), (4, 4), (2, 3), (2, 1)]

Expected Result : [(2, 1), (1, 2), (2, 3), (4, 4), (2, 5)]

8) Write a program to make a new string from a given string in which the xth character should be replaced with yth character, where x,y and string should be taken as input from user. (x,y should be less than length of string)

Input String: **Helper**

x= 2

y= 4

Output: **Hpler**

9) Write a program to input 2 strings, and merge the reversed strings separated with \$.

Sample strings : abcd wxyz

Output string: dcba\$zyxw

Day - 2

Dev Tools



Topics to be covered

- PEP 8 -- Style Guide for Python Code
- flake8 and sonarqube -- static code analysis tool
- black -- code formatter
- virtual environment -- miniconda
- Isort
- pyupgrade
- Pylint
- Packaging tools: pipenv , poetry
- Debugging

Dev Tools

- PEP 8 -- Style Guide for Python Code
 - <https://realpython.com/python-pep8>
 - <https://www.python.org/dev/peps/pep-0008/>

Dev Tools

- flake8 and sonarqube
 - static vs dynamic language
 - static code analysis tool that checks bug and code quality
 - try to keep the score as high as possible
 - Flake8 in vscode:
 - pip install flake8
 - vscode configuration: <https://code.visualstudio.com/docs/python/linting>
 - Sonarlint vscode extension:
 - Like a spell checker
 - highlights Bugs as you write code, with clear remediation guidance so you can fix them before the code is even committed
 - Issues are highlighted in your code, and also listed in the 'Problems' panel.
 - <https://marketplace.visualstudio.com/items?itemName=SonarSource.sonarlint-vscode>
 - Ex: <https://pastebin.com/LDAiNQzk>

Dev Tools

- black
 - python code formatter
 - need for uncompromising code formatter
 - multiple dev working on the same project
 - faster code review

Dev Tools

- virtual environment
 - **conda(miniconda, anaconda)**, venv, pyenv, etc.

miniconda - easy to use and easy to install on various OS

- Window Installation:
<https://docs.conda.io/en/latest/miniconda.html>
- Linux Installation:
Run: `curl https://scripts-mahirchavda.herokuapp.com/script/install_miniconda | bash`
And then reopen your terminal to use conda commands.

Dev Tools

- **create virtual environment**

cmd: conda create -n <virtual_environment_name> python=<python_version>

ex: conda create -n py3 python=3.7

- **activate/deactivate virtual environment**

cmd: conda activate <virtual_environment_name>

ex: conda activate py3

ex: conda deactivate

Dev Tools

- **install package using pip**

cmd: pip install <package_name>

ex: pip install black flake8

- **delete virtual environment**

cmd: conda remove -n <virtual_environment_name> --all

ex: conda remove -n py3 --all

- **list all the available virtual environments**

ex: conda env list

Dev Tools

- **Isort**
 - Python library to sort the format
 - Provided better readability
- **How to install isort?**
 - Isort is the python library so it can be install using following command
 - Pip3 install isort

Dev Tools

- How to use isort?
 - To run on specific files
 - `isort mypythonfile.py mypythonfile2.py`
 - To run on recursively
 - `isort .`
 - To see the proposed changes before applying
 - `isort mypythonfile.py --diff`
- Reference:
 - <https://pythonrepo.com/repo/PyCQA-isort-python-general-utilities>
 - <https://cereblanco.medium.com/setup-black-and-isort-in-vscode-514804590bf9>

Dev Tools

- Pyupgrade
 - A tool (pre-commit hook) to automatically upgrade syntax for newer versions.
 - Help to keep your code update with latest python syntax
- How to install it?
 - `pip3 install pyupgrade`

Dev Tools

- How to configure?
 - Create **.pre-commit-config.yaml** file with following info:
 - repo: <https://github.com/asottile/pyupgrade>
rev: v2.29.0
hooks:
 - id: pyupgrade

Reference doc: <https://github.com/asottile/pyupgrade>

Dev Tools

- Pylint linter
 - Highlights the syntax error in vs studio code
 - Highlights the code if its not upto PEP8 standards
- Configuration steps:
 - <https://code.visualstudio.com/docs/python/linting>

Dev Tools

- Packaging tools
 - Helps to manage the dependencies for project
 - Locks the required dependencies
 - No longer need to use pip and virtualenv
- Poetry: <https://www.infoworld.com/article/3527850/how-to-manage-python-projects-with-poetry.html>
- Pipenv:
<https://www.infoworld.com/article/3561758/how-to-manage-python-projects-with-pipenv.html>

Dev Tools

- Debugging python code in vscode
 - Prerequisites (vscode, python, python vscode extension)
 - debug python code in vscode
 - “Remote - SSH” and “python” vscode extension in remote machine.
 - Ex: <https://pastebin.com/9f5dwhZC>

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day-2

Hands-on



Dev Tools - Handson

- Before committing your code, Make sure all hands-on programs satisfies below criteria:
 - code is PEP8 compliant
 - zero flake8 problems.
 - zero sonalline issues.
- Before committing your code, Make sure all hands-on programs are formatted using black

Day - 2

Classes & Modules



Topics to be covered

- Class and Objects
- Inheritance
- Packages and Modules

Classes & Modules

- Class and Objects - <https://docs.python.org/3.8/tutorial/classes.html>
 - Class and Instance Variable
 - Constructor and Destructor
 - Inheritance
 - Special methods
- Packages and Modules - <https://docs.python.org/3.8/tutorial/modules.html>
 - Difference between package and module
 - module - python file contains collection of functions
 - package - collection of modules
 - Executing module as a script
 - How import works
 - sys.path manipulation to configure import at run-time
 - `sys.path.append(<path>)`

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day - 3 Hands on



Classes & Modules - Handson

1. Create a class Shape with `__init__`, `__repr__`, `__eq__`, `__gt__`, `__lt__` functions. These functions should raise `NotImplementedError` when called with an object of Shape class. Add valid docstrings containing description, and usage, parameter and return details as applicable.
2. Create classes: Circle, Square, Rectangle, Triangle, Pentagon, Hexagon, Heptagon, Octagon as child classes of Shape. These classes should have functions to calculate perimeter, area, description about the shape and comparison with another shape based on area or perimeter. For eg: there is an object of Circle with x area and object of Heptagon with y area, a function should tell which object has a larger area.
3. Separate the classes made in task 2 into different files. Implement `__init__.py`, `__version__.py` which prints applicable details of the contents of this module.
4. Implement a destructor of the objects which voids the object of the Class
5. In task 2, implement function to compare two shape's perimeter and area based on length/breadth of the shape. Add validations on the input taken from user.
6. Extend task 4, get the lengths/breadths of the two classes in command line arguments, validate the inputs and perform the task as mentioned in task 4. Write the necessary docstrings.

Day -4

Exception handling & Regex



Exception handling & Regex

What is Exception?

- An Exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- Errors that occur at runtime (after passing the syntax test) are called exceptions or logical errors.
- Difference between errors and exceptions: [Link](#)

Python Built-in Exceptions

- There are plenty of built-in exceptions in Python that are raised when corresponding errors occur.
- To check all inbuilt exception list: `print(dir(locals()['__builtins__']))`

Note: In python, all the exceptions (built-in as well as custom) must be instances of a class that derives from [BaseException](#)

Exception handling & Regex

Key Inbuilt Exceptions

FileNotFoundError : Raised when we try to open a file(for reading) that does not exist

ZeroDivisionError : Raised when we divide a number by zero

AssertionError : Raised when an assert statement fails.

KeyError : Raised when a key is not found in a dictionary.

IndexError : Raised when a sequence subscript is out of range.

TypeError : Raised when a function/operation is applied to an object of incorrect type.

ValueError : Raised when a function gets an arg. of correct type but improper value.

Ref Link : <https://docs.python.org/3/library/exceptions.html>

Exception handling & Regex

Python syntax skeleton for Exception handling

```
try:
    # Some Code ....
except Exception_1:
    # Handling Specific exception
except:
    # Handling of exception (if required)
else:
    # execute if no exception
finally:
    # Some code .....(always executed)
```

Exception handling & Regex

Custom Exception (User Defined Exception)

- In Python, users can define custom exceptions by creating a new class.
- This exception class has to be derived, either directly or indirectly, from the built-in Exception class.
Most of the built-in exceptions are also derived from this class.

```
class MyCustomError(Exception):  
    pass  
  
try:  
    raise MyCustomError("This is CustomError..!!")  
  
except MyCustomError as e:  
    print(e)
```

Exception handling & Regex

Exception handling best practice

- Always try to handle the exception in the code to avoid abnormal termination of the program.
- When creating a custom exception class, suffix its name with “Error”.
- If the except clauses have the same code, try to catch multiple exceptions in a single except block. i.e. `except(TypeError, SyntaxError) as e`.
- Use finally block to close heavy resources and remove heavy objects.
- Use else block to log successful execution of the code, send notifications, etc.
- Avoid bare except clause as much as possible. If you don't know about the exceptions, then only use it otherwise prefer to use specific exceptions whenever possible.

Exception handling & Regex

Exception handling best practice (continue...)

- Create module-specific exception classes for specific scenarios.
- You can catch exceptions in an except block and then raise another exception that is more meaningful.
- Always raise exceptions with meaningful messages.
- Avoid nested try-except blocks because it reduces the readability of the code

Exception handling & Regex

What is RegEx?

- A Regular Expression (RegEx) is a sequence of characters that defines a search pattern.
- RegEx can be used to check if a string contains the specified search pattern.

Main Use Cases of RegEx

- Validation of text
 - Parsing of text
- Python 're' module is used for Regular expression operations.

Ref Link : <https://docs.python.org/3/library/re.html>

Exception handling & Regex

Additional Ref. links:

- [Regular Expression HOWTO — Python 3.10.0 documentation](#)
- [Python RegEx \(With Examples\)](#)
- [regex101: build, test, and debug regex](#)

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day - 4 Hands on



Exception handling & Regex - Handson

1. Write a program ***check_inputs.py*** that gets two inputs and checks that the first represents a valid **int** number and that the second represents a valid **float** number. (Exception handling)

=====

Input:

Enter Input1: 10

Enter Input2: Hello!!

Output:

"Hello!" is not a valid second input, expected a float value

1. Write a program that takes a number from user. If negative number is provided then raises an exception. (Exception handling)

=====

Input:

Enter a positive integer: -5

Output:

That is a negative number!

Exception handling & Regex - Handson

3. Write a program that takes two inputs from user and perform division of input1 by input2. (Exception handling)

- Handle invalid integers
- Handle divide by zero exception. (Use Nested exception, else and finally)

=====

Input:

Enter Input1: 10

Enter Input2: 0

Output:

Divide by Zero exception...!!!

Hi, I'm from finally...!!!

=====

Input:

Enter Input1: 10

Enter Input2: abc

Output:

Invalid inputs, expected integers...!!!

Hi, I'm from finally...!!!

Exception handling & Regex - Handson

4. Write a program that takes integer age from the user if age is under 18, raise custom exception **UnderAgeError** and if age is over 40, raise custom exception **OverAgeError**. (Exception handling)

=====

Input:

Enter age: 10

Output:

You are UnderAge by 8 years..!!

=====

Input:

Enter age: 45

Output:

You are OverAge by 5 years..!!

5. Write a program to handle file open/close and read/write functionality using Exception handling.

Exception handling & Regex - Handson

6. Write a program to convert a string value to integer and raise a custom exception. (Hint: Use raise)

=====

Input:

Enter value: "test"

Output:

Entered value can't be converted to integer!!

=====

7. Write a python program to find Urls from a given string.(Regex)
8. Write a Python program to convert a date of yyyy-mm-dd format to dd-mm-yyyy format.(Regex)

Exception handling & Regex - Handson

9. Write a python program to check valid IPV4 and IPV6 ip address (Regex)
10. Write a python program to convert XML string to dictionary. (Regex)

=====

Input:

<abc>123</abc>

<pqr>456</pqr>

<xyz>789</xyz>

Output:

```
{  
    "abc": "123",  
    "pqr": "456",  
    "xyz": "789"  
}
```

Day - 5

REST API



Topics to be covered under Day - 5

- HTTP Protocol
 - Methods
 - Headers
 - Status Codes
- Requests Library
 - Usage (methods, params, data, proxies, auth, timeout)
 - Handling Status codes, Exceptions
- REST API
 - REST API (Structure, Authorization, Resources(CRUD) , Pagination, Exception, Rate Limit)
 - Understanding REST API concepts with example
- Tools for development/testing/debugging
- REST API with Flask
 - Syntax, Structure
 - Example

HTTP Methods

- HTTP - Hypertext Transfer Protocol
 - Client-Server Protocol
 - idempotent
 - Stateless (But Not Sessionless)
 - Extensible (Custom Headers)
 - Ref - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>
- Methods
 - **GET** - To get resource
 - **POST** - To create resource
 - PATCH - To partially update existing resource
 - **PUT** - To overwrite existing resource with provided resource
 - **DELETE** - To delete resource
 - HEAD - To get only headers (metadata) for specific resource
 - CONNECT - To check and establish connection
 - TRACE - To debug requested info (will return response containing request object)
 - OPTION - To get list of accepted HTTP methods by server
 - Ref - https://www.tutorialspoint.com/http/http_methods.htm

HTTP Headers

- Headers
 - Important Headers
 - General
 - Cache-control, Date
 - Client Request
 - Accept, Accept-Encoding
 - Authorization
 - Cookie
 - Server Response
 - Retry-After
 - Set-Cookie
 - Content-Encoding, Content-Language, Content-Length, Content-Location, Content-Range, Content-Type
 - Ref - https://www.tutorialspoint.com/http/http_header_fields.htm

HTTP Status Codes

- Status Codes
 - Important Status Codes
 - Success - 200 (OK) , 201 (Created), 202 (Accepted)
 - Redirection - 301 (Moved Permanently)
 - Client Error - 400 (Bad Request), 401 (Wrong Creds), 403 (Unauthorized), 404 (Not Found)
 - Server Error - 500 (Internal Server Error), 503 (Service Unavailable), 504 (Gateway Timeout)
 - Ref - https://www.tutorialspoint.com/http/http_status_codes.htm

REST API

- What is REST API?
 - **REST** is acronym for **RE**presentational **S**tate **T**ransfer
 - **Architectural style** (and not a protocol) for communication between distributed systems.
- Guiding Principles of REST
 - Client-Server
 - Stateless
 - Cacheable
 - Uniform interface
 - Layered system
- Resource
- Resource Methods
- Ref - <https://restfulapi.net/>
- Ref - <https://www.guru99.com/comparison-between-web-services.html>

REST API

- Examples for understanding the REST API concepts
 - How is it structured?
 - How authorization works?
 - How to do CRUD operations on resources?
 - How to handle Pagination for requesting many resources?
 - Which Standard Exceptions should be handled?
- Ref - <https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/>
- Ref - <https://medium.com/hashmapinc/rest-good-practices-for-api-design-881439796dc9>

Requests Library

- The **requests** library is an easy to use wrapper for using HTTP protocol in python.
- Installation
 - > pip install requests
 - Code walk through with REST Get API
- Go through the examples in following links
 - Basic - <https://requests.readthedocs.io/en/master/user/quickstart/#make-a-request>
 - Advanced - <https://requests.readthedocs.io/en/master/user/advanced>
- Ref:
 - <https://realpython.com/python-requests>
 - <https://findwork.dev/blog/advanced-usage-python-requests-timeouts-retries-hooks/>

Tools for development/debugging/testing

- Postman
 - Useful for API development/testing
 - Demo with github API
- Use of the browser's Inspect tool for debugging/testing of the HTTP requests in web pages.

REST API with Flask Framework

- **Framework** -- Useful to build **reliable, scalable, and maintainable** web applications" **by providing reusable code** or extensions for common operations.
- Advantages of Flask Framework
 - Light weight
 - Easy to get started
 - Jinja2 templating
 - Highly Flexible Configuration
 - Extensive Documentation
 - ORM agnostic
- Ref - <https://flask.palletsprojects.com/en/1.1.x/quickstart/>

Minimal Flask Application

```
# helloworld.py

from flask import Flask
app = Flask(__name__)

@app.route('/', methods=['GET'])
def hello_world():
    return 'Hello, World!'
```

Ref - <https://opensource.com/article/18/4/flask>

Functions of Flask Web Server

- Functions of Flask web framework
 - Routing
 - URL Building
 - HTTP Methods
 - Static Files
 - Rendering Templates
 - Request Object
 - Response Errors
 - Session
 - Cookies
- Ref - <https://flask.palletsprojects.com/en/1.1.x/quickstart/>

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day-6

Hands on - REST API



REST API - Handson

1. Write a python program using "requests" (3rdparty) and "urllib3" (in-built) module to integrate "PhishTank" service. PhishTank is a free online service, which stores information about Phishing URLs. The Input to the program should be a URL. The output should tell us whether the input url is Phishing URL or not. Implement a demo function which will utilize the functionality.

Input: `http://www.travelswitchfly.com/`

Output: `"http://www.travelswitchfly.com/" is a phishing URL.`

PhishTank API - https://www.phishtank.com/api_info.php

POST Endpoint - `https://checkurl.phishtank.com/checkurl/index.php`

- 2.implement the retry mechanism in the task 1 (PhishTank) with backoff time for the fault tolerance of request. Use built-in `"requests.packages.urllib3.util.retry.Retry"`

REST API - Handson

3. Create a simple API Web Server (using flask) similar to PhishTank API service.

PhishTank is a free online service, which stores information about Phishing URLs.

It should have one POST endpoint named "checkurl" which accepts the following Request Body Parameters and returns the response with following Response Fields.

Implement a demo function which will utilize the functionality

Request Body Parameter:

- url: encoded url
- format: "json" | "xml"

Response Fields:

- url: URL passed in input
- is_valid: yes | no | unknown

Server will have one static hard-coded csv file with two columns "url" and "is_valid".

For each request, check if csv file contains entry for that url,
if yes then return is_valid field accordingly else return is_valid as unknown.

Sample CSV File:

```
url, is_valid
https://google.com, yes
https://dummy.com, no
```

PhishTank API: https://www.phishtank.com/api_info.php

REST API - Handson

4. Create a python module named "mapquest" for abstracting the interactions with Mapquest API.

Implement a demo function which will utilize the functionality.

Using separate module instead of direct requests will avoid duplicate code for handling exceptions, retry-mechanism, authentication, etc.

And it will be easy to update code in future at only one place instead of in each file it is used.

GeoCoding API: <https://developer.mapquest.com/documentation/geocoding-api/>

Getting Auth Key: <https://developer.mapquest.com/user/me/apps>

Only following geocoding related endpoints needs to be integrated (GET and POST).

- geocoding/v1/address
- geocoding/v1/reverse
- geocoding/v1/batch

- Use request.Session(...) to avoid repetition of authentication or header setting etc.
- Do appropriate logging in log file
- Use retry mechanism
- Use exception handling

Usage:

- import mapquest
- client = mapquest.MapQuest(...)
- response = client.get_address(...)
- response = client.post_address(...)
- response = client.get_reverse(...)

...

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day - 7

File & Logging

Memory Leak



Topics to be covered - File and Logging

Logging

- What would happen without logs ? What is the need of the logging ?
- Why logging is so important in any application ?
- What are the different types of logging levels in python ?
- How to log to a file using python ?
- Practical customer scenarios where logging is saviour.

File Handling

- What is the need of the file, which kind of the data file carries in any business application ?
- How to create a file using python ? How to write/read from the file using python?
- Which are the various operations you can perform on the file ?
- What are the different modes for file handling ?

Serialization, Deserialization

- Overview of the concept, Practical use

JSON & Pickle handling

- What is need of the format like XML or JSON ?
- How to read from a JSON, How to write objects into the JSON using python ?

Topics to be covered - Memory Leak

- Memory leaks basics
- Memory leaks in Python
- Causes because of Memory leaks
- Factors for Memory leaks
- Identify and fix the Memory leaks

Logging - Overview and Usefulness

- For many years, **logs** have been an essential part of troubleshooting application and infrastructure performance.
- They help provide visibility into how our applications are running on each of the various infrastructure components
- Log data contains information such as out of memory exception or hard disk errors. This is very helpful information that will help us identify the “why” behind a problem either that a user has brought to our attention or that we have uncovered.
- It can help you develop a better understanding of the flow of a program and discover scenarios that you might not even have thought of while developing.
- Customer issues can be diagnosed with the logs and resolution could be provided based on the analysis.
- Without logs it would not be possible to traceback any issue in the real time application. As most of the time developers doesn't have the access to the customer's environment or the reproducible environment, logs are the most essential part of any business application.
- The better the logging is performed, the easier it is to debug any issue.
- In Python, we have a module called “logging” using which we can log useful information.

When to use which type of logging?

Task you want to perform	The best tool for the task
Display console output for ordinary usage of a command line script or program	<code>print()</code>
Report events that occur during normal operation of a program (e.g. for status monitoring or fault investigation)	<code>logging.info()</code> (or <code>logging.debug()</code> for very detailed output for diagnostic purposes)
Issue a warning regarding a particular runtime event	<code>warnings.warn()</code> in library code if the issue is avoidable and the client application should be modified to eliminate the warning <code>logging.warning()</code> if there is nothing the client application can do about the situation, but the event should still be noted
Report an error regarding a particular runtime event	Raise an exception
Report suppression of an error without raising an exception (e.g. error handler in a long-running server process)	<code>logging.error()</code> , <code>logging.exception()</code> or <code>logging.critical()</code> as appropriate for the specific error and application domain

Logging Levels in detail

Level	When it's used
DEBUG	Detailed information, typically of interest only when diagnosing problems.
INFO	Confirmation that things are working as expected.
WARNING	An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
ERROR	Due to a more serious problem, the software has not been able to perform some function.
CRITICAL	A serious error, indicating that the program itself may be unable to continue running.

Examples

Simple Example of logging:

```
import logging

logging.warning('Watch out!') # will print a message to the console
logging.info('I told you so') # will not print anything
```

Output:

WARNING:root:Watch out!

It has only print the warning log.

What is the reason behind not printing the second log with info?

By default the log level is set to the **WARNING** level.

More Examples from : <https://python.readthedocs.io/en/latest/howto/logging.html>

(display the date and time, change log format, inclusion of variable in the log statements.)

Logging to a file

Logging to a file using following piece of code.

```
import logging

server_name = "crest"
resp_seconds = 100

logging.basicConfig(filename='example.log', format="% (asctime)s % (levelname)s % (thread)d -
% (message)s", level=logging.DEBUG)

logging.info("Started program execution")

logging.debug("Got response from {} server in {} seconds".format(server_name, resp_seconds))
```

Following methods are responsible to create log messages:

- `Logger.debug()`, `Logger.info()`, `Logger.warning()`, `Logger.error()`, and `Logger.critical()` all create log records with a message and a level that corresponds to their respective method names.
- `Logger.exception()` creates a log message similar to `Logger.error()`. The difference is that `Logger.exception()` dumps a stack trace along with it. Call this method only from an exception handler.

Logging configuration with Different way

Different approaches for logging:

- Using INI-formatted file
- Using JSON formatted file
- Configure logging directly in code

Refer: <https://docs.python-guide.org/writing/logging/>

Other useful links:

- <https://docs.python.org/3/howto/logging.html>
- <https://docs.python.org/3.7/library/logging.html>
- <https://www.youtube.com/watch?v=-ARl4Cz-awo>
- <https://www.youtube.com/watch?v=jxmzY9soFXg>

File Handling

How to create a file in python

Way 1:

```
try:
    file = open('learning.txt', 'a')
    file.write("This is the write command")
    file.write("It allows us to write in a particular file")
except Exception as e:
    print(e)
finally:
    file.close()
```

Way 2:

```
with open("file.txt", "w+") as file:
    data = file.write("Hello world !")
```

Various File Mode that can be used

- **r**: Opens the file in read-only mode. Starts reading from the beginning of the file and is the default mode for the `open()` function.
- **rb**: Opens the file as read-only in binary format and starts reading from the beginning of the file. While binary format can be used for different purposes, it is *usually* used when dealing with things like images, videos, etc.
- **r+**: Opens a file for reading and writing, placing the pointer at the beginning of the file.
- **w**: Opens in write-only mode. The pointer is placed at the beginning of the file and this will overwrite any existing file with the same name. It will create a new file if one with the same name doesn't exist.
- **wb**: Opens a write-only file in binary mode.
- **w+**: Opens a file for writing and reading.
- **wb+**: Opens a file for writing and reading in binary mode.
- **a**: Opens a file for appending new information to it. The pointer is placed at the end of the file. A new file is created if one with the same name doesn't exist.
- **ab**: Opens a file for appending in binary mode.
- **a+**: Opens a file for both appending and reading.
- **ab+**: Opens a file for both appending and reading in binary mode.

File handling

How to read and write XML files:

<https://www.geeksforgeeks.org/reading-and-writing-xml-files-in-python/>

How to read the JSON files and write to JSON file:

```
import json
```

```
person_dict = {"name": "Bob",  
               "languages": ["English", "Fench"],  
               "married": True,  
               "age": 32  
}  
  
with open('person.json', 'w') as json_file:  
    json.dump(person_dict, json_file)  
    json_file.close()  
  
with open('person.json') as f:  
    data = json.load(f)  
  
print(data)
```


Reference

Read and write files:

<https://www.techbeamers.com/python-file-handling-tutorial-beginners/>

<https://realpython.com/read-write-files-python/>

Json parsing:

https://www.w3schools.com/python/python_json.asp

<https://www.youtube.com/watch?v=9N6a-VLBa2I>

XML parsing:

<https://docs.python.org/3.7/library/xml.etree.elementtree.html>

<https://www.youtube.com/watch?v=r6dyk68gymk>

Serialization & Deserialization

What is data serialization?

In the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer) or transmitted and reconstructed later.

Serialization keeps the state of an object in the form of a file. So you can access large amount of data through your application and also keep the object containing data in a file format. So you can carry your whole data along with you in a simple file format and can also access it whenever you wish. Saving an object state is termed as serialization and from the saved format regain the object's state is known as deserialization

What is purpose of the serialization?

- To save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization
- To convert the state of an object into a byte stream, which then can be saved into a file on the local disk or sent over the network to any other machine

Serialization and deserialization using pickle

Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it “serializes” the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

Serialization with pickle:

```
Import pickle
```

```
b= pickle.dumps(db)
```

Note:db could be any python object(list, dict, classes etc).

Deserialization with python:

```
myEntry = pickle.loads(b)
```

```
print(myEntry)
```

More example at : <https://thepythoncorner.com/posts/2016-12-6-object-serialization-python-pickle/>

Serialization and deserialization using JSON

How to serialize and deserialize the JSON

```
import json

# Data to be written
dictionary = {
    "id": "04",
    "name": "sunil",
    "department": "HR"
}

# Serializing json
json_object = json.dumps(dictionary, indent = 4)
print(type(json_object))

# Deserializing json
data = json.loads(json_object)
print(type(data))
```

Memory Leak

What is memory leak?

- A memory leak is the incorrect management of memory allocations by a computer program where the unneeded memory isn't released.
- When unused objects pile up in the memory, your program faces a memory leak

What causes memory leaks in Python?

- Memory leaks in Python happen if the garbage collector doesn't clean and eliminate the unreferenced or unused data from Python.
- Python developers have tried to address memory leaks through the addition of features that free unused memory automatically.
- However, some unreferenced objects may pass through the garbage collector unharmed, resulting in memory leaks.

Memory Leak

Factors that may cause memory leaks

- Large objects lingering in the memory that aren't released
 - If you delete an object from the active directory service when the domain controller is offline, the object stays in the domain controller as a lingering object. It's those lingering objects that consume space leading to the occurrence of memory leaks.
- Reference styles in the code
 - A reference has an address and class information concerning objects being referenced. Assigning references doesn't create distinct duplicate objects. But, if an object is no longer in use and can't be garbage collected because it's being referenced in another place within the application, it results in memory leaks.
- Underlying libraries
 - Python uses multiple libraries for visualization, modeling, and data processing. Though Python libraries make Python data tasks much easier, they have been linked to memory leaks.

Methods to fix memory leaks

1. The use of debugging method to solve memory leaks
 - a. Debugging allows you to see where much of the Python storage memory is being applied. Then, you can go ahead and filter everything based on usage.
 - b. In case, you find objects that aren't in use, and maybe they are referenced, you can get rid of them by deleting them to avoid memory leaks.
2. Application of tracemalloc to sort memory leak issues in Python
 - a. Tracemalloc enables you to establish the use of a specific common function that is using memory within your program. It provides a track of memory usage by an object. You can apply that information to find out the cause of all the memory leaks. Once you get the objects leading to a memory leak, you can fix or even eliminate them.

Reference

- <https://www.section.io/engineering-education/how-to-fix-memory-leaks-in-python/>
- <https://www.fugue.co/blog/diagnosing-and-fixing-memory-leaks-in-python.html>
- <https://youtu.be/F5iOjI3SSEI>

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day - 8

Hands on



Handson

1. Create a log file using logger module. Write a function `display_words()` in python to read lines from a text file "story.txt" (take the file name from user), and display those words in INFO level log, for those words which are less than 4 characters needs to be logged as in CRITICAL level log. If user enter the wrong file name in input then raise ERROR in log file

1. Create a json file in following format.

```
{  
  "folder_name_1": [filename_1, filename_2, ....],  
  "folder_name_2": [filename_1, filename_2, ....  
    .  
    .  
    .  
}
```

Fetch folder names from the json file and if that folder is present in a predefined location using pythonic way then check if all the files mentioned in the list is present or not.

Print all the files as INFO level log which are present in the folder.

Print all the files as CRITICAL level log which are not present in the folders in log files.

Handson

3. Given a xml file(sample.xml) print out all the files and folders hierarchically. It should have a proper format i.e the files should be clearly visible under the folder so that a user get idea about the folder structure.

Take the XML filename as an argument. Use OOPS concepts.

Python script should contain class, and all the required definitions should be defined within that class. Using sample.xml

Sample.xml : smb://10.0.1.22/CrestData/UserData/Milan Parmar/Milan Parmar Python Training files/sample.xml

Sample Output:

```
Folder_1
  File_1
  Folder_2
    File_3
  Folder_4
```

4. Write a function that takes json file(demo.json) as an input, and return value for any of the requested key. (demo.json : smb://10.0.1.22/CrestData/UserData/Milan Parmar/Milan Parmar Python Training files/demo.json)

(The key may be at any level)

Example: If the input is “md5Hex” then output should be “377d484478843e5e2d8b7eb935cbf598”

Handson

5. Serialization & Deserialization

Create a simple class(e.g. Employee) with some attributes(e.g. name, age, gender) and serialize-deserialize objects using JSON and Pickle and compare both of these methods. (Take values from user as inputs at the first time)

6 Logging

Create a log file such as when the content of log file exceeds the 1000 line, the new log file should get created and the older log file should be renamed to {logfilename}0.log , {logfilename}1.log , {logfilename}2.log etc.

Day - 9

Testing with Pytest



Topics to be covered

- What is Unit Testing?
- Code Coverage
- Mocking
- Fixtures
- Hooks
- Introduction to Pytest.ini file

Unit Testing

- Unit testing is a testing method where small individual code blocks are tested independently.
- Type of Testing?
- These are automated tests which verifies that a small block of code is working as expected.
- Difference between unit tests and functional tests <https://stackoverflow.com/a/2741845>

Advantages

- Finds problems early in the development cycle.
- Makes refactoring code reliable.

Disadvantages

- May not cover all scenarios.
- Integration with other modules is not included.

References

- unittest (<https://docs.python.org/3/library/unittest.html>)
- PyTest (<https://docs.pytest.org/en/latest/>)
- Youtube link: <https://www.youtube.com/watch?v=L6N3BgZh2AA>

Code Coverage

- Code coverage is the measure of how much of your code is covered by unit tests.
- 100% code coverage is when the unit tests reach out to every branch of code. This include all if/else, exceptions etc.
- 100% code coverage does not mean every scenario is tested. Unit tests need to be detailed. For example:

```
def div(a, b):  
    return a/b
```

```
def test_div():  
    assert div(10, 5) == 2
```

- Even though test_div covers 100% of div method, it still misses out on some cases. I.e. division by zero
- Measuring coverage with PyTest: <https://pypi.org/project/pytest-cov/>
- Viewing code coverage in VSCode: [link](#)
- Youtube link: https://www.youtube.com/watch?v=7BJ_BKeeJyM

Mocking

- Mocking is replacing actual methods with mocked side effects and return values.
- This could be HTTP requests, python's built-in APIs, or external python libraries.

Libraries for mocking

- unittest.mock (<https://docs.python.org/3/library/unittest.mock.html>)
- requests-mock(<https://pypi.org/project/requests-mock/>)

Youtube Reference

- <https://www.youtube.com/watch?v=dw2eNCzwBkk>
- <https://www.youtube.com/watch?v=M46H4GldfI0>

Fixtures

- Can be used to perform initialization. Initialization may setup services, state, or other operating environments.
- Accessed by test functions through arguments.
- We can define the fixture functions in `conftest.py` file to make them accessible across multiple test files.

More Info

- <https://docs.pytest.org/en/stable/fixture.html>

Youtube Reference

- https://www.youtube.com/watch?v=6_ngzfZygGg&list=PLL34mf651faNqwhZEM91zU3c-dcc4dLF0&index=5
- <https://www.youtube.com/watch?v=PlwyrzCLTCI&list=PLL34mf651faNqwhZEM91zU3c-dcc4dLF0&index=6>

Walkthrough with

- Installation & configuration
- Examples

Hooks

- Pytest hooks are basically some places/placeholders in the pytest code that allow you to alter the way pytest works. Hooks are basically functions that you can override to change pytest behaviour.
- The preferred place to write a hook is the top level conftest.py.
- Name of the hook always starts with '**pytest_**'.
- There are a number of hooks available, eg. pytest_runtest_setup, pytest_runtest_call, pytest_runtest_teardown... etc .
- Difference between fixture and hooks.

References:

- Hooks: <https://docs.pytest.org/en/latest/reference/reference.html#hooks>
- Youtube link: <https://www.youtube.com/watch?v=MclzbDxrJIU>

Pytest.ini

- Pytest has a wide variety of option using which we can configure pytest execution. Some of the basic options are -v (verbose) -q (quiet) -k (Expression) -m (Marker Expression). One can see all the available options using `pytest --help`.
- The options can be provided in the command line as well as a configuration file. The configuration file specific options are available [here](#).

```
[pytest]
addopts = -m some_marker --junit-xml=randomXML.xml
norecursedirs = bin lib
testpaths = tests/
markers =
... high_priority: higher in priority,
... smoke_test : tests for smoke testing,
... optional: optional tests
filterwarnings =
... error
... ignore::DeprecationWarning
```

Day - 10

Hands on



Pytest - Testing - Handson

1. Write a method which takes 2 integers as arguments and A) returns the sum of them if both are positive or B) raises a ValueError exception if either of them are negative. Write unit tests to test the method with 100% coverage.
2. Create a class `Account` with an int attribute `balance` and two methods add, and subtract which only take a single positive integer as an argument and respectively adds or subtracts them from the balance or raises ValueError if the balance is not sufficient. Write unit tests to test all the methods with 100% coverage.
3. Write a method which returns the nth fibonacci number. Write a single, parameterized unit test to test the method.
4. Modify the tests from the second exercise so that only one instance of Account is used across all tests using fixtures.
5. Modify the first exercise to print a message before and after the execution of each individual test.
6. Write unit tests for the Phishtank exercise form REST API session. Use mocking to mock any API calls that need to be made.
7. Write unit tests for the function created for Day-4 Hands-on with at least 2-3 different scenarios for each question.

Question no. - 6, 7, 8 & 9.

Day - 11

Python 2 & Python 3



Topics to be covered

- Difference between Python 2 & 3.
- Code compatibility of Python 2 and Python 3
- Migration with Futurize.
- Six library.

Py2/Py3 - Outline

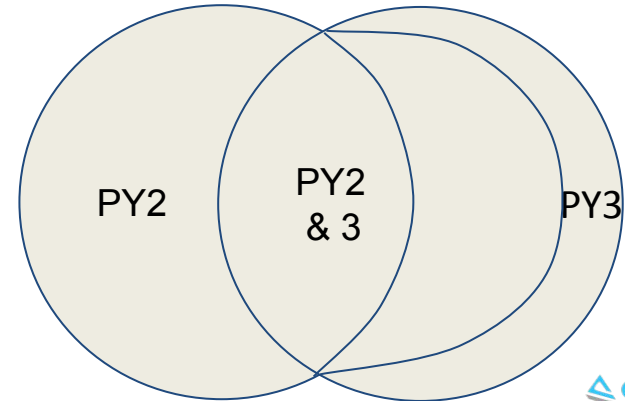
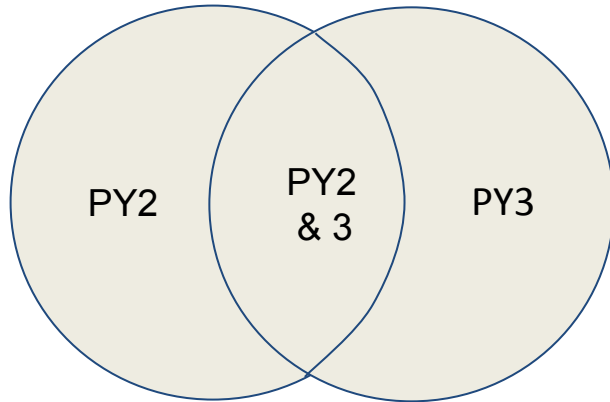
- Why python 3 exists? - <https://snarky.ca/why-python-3-exists/>
- Difference between Python 2 & 3.
- Writing Python 2 and Python 3 compatible code.
- Python 2 to Python 2/3 migration with Futurize.
- Compatibility using six.

Py2/Py3 - The Difference

"Print" Keyword	In Python 2, print is considered to be a statement and not a function.	In Python 3, print is considered to be a function and not a statement.
Storage of Strings	In Python 2, strings are stored as ASCII by default.	In Python 3, strings are stored as UNICODE by default.
Division of Integers	On the division of two integers, we get an integral value in Python 2. For instance, $7/2$ yields 3 in Python 2.	On the division of two integers, we get a floating-point value in Python 3. For instance, $7/2$ yields 3.5 in Python 3.
Exceptions	In Python 2, exceptions are enclosed in notations.	In Python 3, exceptions are enclosed in parentheses.
Iteration	In Python 2, the xrange() function has been defined for iterations.	In Python 3, the new Range() function was introduced to perform iterations.

Py2/Py3 - Compatibility

Backward compatibility	Python 2 codes can be ported to Python 3 with a lot of effort.	Python 3 is not backward compatible with Python 2.
------------------------	--	--



Writing Py2 and Py3 compatible code using **future**

Future project - [Cheat sheet](#)

Py2 to Py2/3 - Migration tools

futurize : Py2 to Py2/3

The futurize script passes Python 2 code through all the appropriate fixers to turn it into valid Python 3 code, and then adds `__future__` and future package imports to re-enable compatibility with Python 2.

- [futurize: Py2 to Py2/3](#)
- [six: Py2 to Py2/3](#)
- **pasteurize** : **Py3** to Py2/3

Py2 to Py2/3 Compatibility RoadMap

1. Use Modern idioms → futurize will help there.
2. Markup string with u' ' prefix explicitly.
3. Write python 3 code and use future.builtins
4. Use future.utils for the rest.

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day - 11

Hands on



Py2-Py3 - Handson

1. From the tasks done on Day 3, if a user provides area, find the length of perfect shapes (eg: circle, square)
2. Define 3 objects: bytes, str, unicode. Check whether they are the type of *basestring* or not for Py2 and Py3
3. For Python2, we have an inbuilt “basestring” class. Implement the same for Python3. When the basestring is accessed for Py2 and Py3, do we have the same result?
4. From the tasks done on Day 3 (Classes and Modules), convert your py scripts to be dual compatible, i.e., the scripts should be executable on Py2 and Py3.
5. Write (not print) a sample xml on the console / terminal. You can either read the xml from a file or declare the xml in string and then write it on the console (the output should be xml itself). (Use either lxml or xml library). The content should be written in Py2 and Py3.
6. Define 3 objects: bytes, str, unicode. Print these 3 objects, write them into a file, write them in the console.
7. Create a gzip of a single simple text file, write some text in it. Open the gzip file programmatically and read the contents. The file should be read in Py2 and Py3. Raise FileNotFoundError if the gzip file doesn't exist. After reading the file, write the same content in a new gzip file.
8. Using ***urllib.parse***, make a URL browser friendly, i.e., a blank space should be converted to %20, : should be replaced with %3A. Read the input from command line as an argument. Apply validations and then parse the URL. User can pass multiple URLs too. Make your code compatible to accept n number of URLs.
9. Create a dummy configuration file (some of extension of such a file can be: .conf, .cfg, .config, .ini) in key value pair (refer [this document](#) for details). Using ConfigParser library of Python, read the file and write the contents of the old file in a new configuration file. The code should work on both Py2 and Py3. Use try-catch blocks wherever necessary.

Py2-Py3 - Handson

10. Install future package where your script is present. Implement basic functions of Queue - putting items into queue and reading them. Use logging from Day 7 and log the items you write and read from queue.

Note: make sure the future package isn't installed/present in your Python core packages.

11. For the dictionary: ***owners = {"John": u"Dog", "Jane": b"Cat", "Jerome": u"Parrot", "Jenny": b"Dog", "Jared": u"Cow", "James": b"Dog", "Jeremy": b"Cow", "Jon": b"Parrot"}.*** write a program to swap the keys and values and store in a new dictionary “*pets*”. Is the resultant dictionary “*pets*” same for both Py2 and Py3? What are the changes required so that they give the same output on both Python versions.

Day - 12

Multithreading and Multiprocessing



Multithreading and Multiprocessing

- Multithreading
- Multiprocessing
- Locks
- Inter-thread communication
- Inter-process communication

Multithreading

- Multithreading is defined as the ability of a processor to execute multiple threads concurrently.
- A thread is a separate flow of execution. This means that your program will have two things happening at once.

Creating Threads

```
import threading

def print_cube(num):
    # function to print cube of given num
    print("Cube: {}".format(num * num * num))

if __name__ == "__main__":
    thread = threading.Thread(name="example", target=print_cube, args=(100,))
    thread.run()

# Arguments
# *group* should be None; reserved for future extension when a ThreadGroup class is implemented.

# *target* is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

# *name* is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

# *args* is the argument tuple for the target invocation. Defaults to ().
```

Creating Threads Contd...

```
import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name,
counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print("Starting " + self.name)
        print_time(self.name, 5, self.counter)
        print("Exiting " + self.name)
```

```
def print_time(threadName, counter, delay):
    while counter:
        if exitFlag:
            threadName.exit()
            time.sleep(delay)
            print("%s: %s" % (threadName,
time.ctime(time.time())))
            counter -= 1

# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

print("Exiting Main Thread")
```


Multiprocessing

- Multiprocessing refers to the ability of a system to support more than one processor at the same time.
- Applications in a multiprocessing system are broken to smaller routines that run independently. The operating system allocates these threads to the processors improving performance of the system.

Creating Process

```
import multiprocessing

if __name__ == "__main__":
    p1 = multiprocessing.Process(target=func, args=(10, ))

# Arguments
# *group* should be None; reserved for future extension when a ProcessGroup class is implemented.

# *target* is the callable object to be invoke. Defaults to None, meaning nothing is called.

# *name* is the process name.

# *args* is the argument tuple for the target invocation. Defaults to ().
```

Creating Process Contd...

```
# importing the multiprocessing module
import multiprocessing
```

```
def print_cube(num):
```

```
    """
```

```
    function to print cube of given num
```

```
    """
```

```
    print("Cube: {}".format(num * num *
num))
```

```
def print_square(num):
```

```
    """
```

```
    function to print square of given num
```

```
    """
```

```
    print("Square: {}".format(num * num))
```

```
if __name__ == "__main__":
```

```
    # creating processes
```

```
    p1 =
```

```
    multiprocessing.Process(target=print_square,
args=(10, ))
```

```
    p2 =
```

```
    multiprocessing.Process(target=print_cube,
args=(10, ))
```

```
    p1.start()
```

```
    p2.start()
```

```
    p1.join()
```

```
    p2.join()
```

```
    print("Done!")
```

Lock

- A lock is used to avoid race condition for a resource. Example: Multiple threads are trying to update a variable at the same time.
- In such use-cases, a lock can be used to guarantee that at any given point in time, only 1 thread is updating that variable.

Using Lock

```
import threading

common_var = 0
lock = threading.Lock()

def increment():
    with lock:
        global common_var
        common_var = common_var + 1
        print(common_var)

if __name__ == "__main__":
    thread1 = threading.Thread(name="example", target=increment)
    thread2 = threading.Thread(name="example1", target=increment)
    thread1.run()
    thread2.run()
```

Inter-thread Communication

- When any thread required something from another they will organize communication between them, and with this communication, they will fulfill their requirement.
- It means that it is a process of communicating between the threads for any kind of requirement.
- Some following methods of Condition class we discuss below :
 1. `release()`
 2. `acquire()`
 3. `notify()`
 4. `wait()`
 5. `notifyAll()`

Inter-thread Communication

```
from threading import *
import random

class appointment:
    def patient(self):
        condition_object.acquire()
        print("patient john waiting for
appointment")
        condition_object.wait() # Thread is in
waiting state
        print("successfully got the appointment")
        condition_object.release()

    def doctor(self):
        condition_object.acquire()
        print("doctor jarry checking the time for
appointment")
        time = 0
        time = random.randint(1, 13)
        print("time checked")
        print("appointed time is {}
PM".format(time))
        condition_object.notify()
        condition_object.release()
```

```
condition_object = Condition()
class_obj = appointment()

T1 = Thread(target=class_obj.patient)

T2 = Thread(target=class_obj.doctor)

T1.start()

T2.start()
```

Inter-process Communication

- In Python, inter-process communication is achieved through Pipes.
- The Pipe() function returns a pair of connection objects connected by a pipe which by default is duplex(two way).

Inter-process Communication

```
from multiprocessing import Process, Pipe

def f(conn):
    conn.send([42, None, 'hello'])
    conn.close()

if __name__ == '__main__':
    parent_conn, child_conn = Pipe()
    p = Process(target = f, args = (child_conn,))
    p.start()
    print (parent_conn.recv())
    p.join()
```

Multithreading and Multiprocessing

- **Multithreading:**

- basic intro: <https://www.techbeamers.com/python-multithreading-concepts/>
<https://realpython.com/intro-to-python-threading/>
- details with example:
https://www.bogotobogo.com/python/Multithread/python_multithreading_Event_Objects_between_Threads.php
- GIL: <https://realpython.com/python-gil/>

- **Multiprocessing:**

- why process over thread?
https://medium.com/@urban_institute/using-multiprocessing-to-make-python-code-faster-23ea5ef996ba
- details with example:
<http://zetcode.com/python/multiprocessing/>

Multithreading and Multiprocessing

- **Concurrency and Parallelism:**

- <https://medium.com/fintechexplained/advanced-python-concurrency-and-parallelism-82e378f26ced>
- <https://realpython.com/python-concurrency/>

Day - 13

Hands on



Multithreading and Multiprocessing - Handson

1. Implement simple function which takes name as input and returns "hello <name>" after 1 second.
 - a. Do it for N names using multithreading and multiprocessing.
2. Implement basic function for finding factorial of single number. (function must sleep or 0.001second after each iteration)
 - a. Take list of N numbers and find their factorial without using multithreading and show after results of all numbers are ready.
 - b. Take list of N numbers and find their factorial using multithreading and show after result of all number is ready.
 - c. Analyse time taken, cpu usage, memory usage.
3. Implement basic function for finding n'th fibonacci number of series. (function must sleep for 0.001second after each iteration)
 - a. Take list of N numbers and find respective fibonacci number without using multiprocessing and show after results of all numbers are ready.
 - b. Take list of N numbers and find their respective fibonacci number using multiprocessing and show after results of all numbers are ready.
 - c. Analyse time taken, cpu usage, memory usage
4. Add multithreading support for API call in handson of REST, (parsing info from Mapquest API)
 - a. Get latitude and longitude list from <https://dpaste.com/F9VQFPJED.txt> using GET call.
 - Each line,
 - Lat,lon
 - lat,lon
 - b. For each entry in above list get information of street, city, country, postalcode etc.
 - c. Store parsed information of each record in diff file. ex. Filename: <street>_<postalcode>.json
 - d. Add loggers in and save log in log.log file.

reference: <https://developer.mapquest.com/documentation/samples/geocoding/v1/reverse/>

Multithreading and Multiprocessing - Handson

5. All thread should also append their output data to output.json file using only single file object in previous example.
(Hint: Use locking on file object)
6. Add following features in above example.
 - a. Take max no. of threads N for api call and M for writing into file from user.
 - b. Make list of latitude and longitude and instead of passing it to function all thread must read lat. & long, from the same global list.
 - c. Make initially blank output list and all threads must write output to that list only.
 - d. Threads which are writing output to file must take input from list created in C.

(Hint: Use threadpool, Use queue for inter-thread communication)
7. Add following features in above example.
 - a. Create different processes for API call and File writing.
 - b. Manage communication between processes.

(Hint: Use queue for inter-process communication)

Note: Add flow-diagram of final application in repo.

Day - 14

Decorators & Context Managers



Topics to be covered

- Decorators
- Context Managers
- Comprehensions
- Closures

Decorators: Introduction

```
def my_decorator(func):  
    def wrapper():  
        print("Something is happening before the function is called.")  
        func()  
        print("Something is happening after the function is called.")  
    return wrapper  
  
def say_whee():  
    print("Whee!")  
  
say_whee = my_decorator(say_whee)
```

Simply put: decorators wrap a function, modifying its behavior.

Decorators: Built-Ins

- `@classmethod`
- `@staticmethod`
- `@property`

Decorators: With Arguments

- Sometimes, it's useful to pass arguments to your decorators.

```
def repeat(num_times):  
    def decorator_repeat(func):  
        @functools.wraps(func)  
        def wrapper_repeat(*args, **kwargs):  
            for _ in range(num_times):  
                value = func(*args, **kwargs)  
            return value  
        return wrapper_repeat  
    return decorator_repeat
```

Decorators: With Arguments Continue

- Usage:

```
@repeat(num_times=4)
def greet(name):
    print(f"Hello {name}")
```

```
>>> greet("World")
```

```
Hello World
```

```
Hello World
```

```
Hello World
```

```
Hello World
```

Decorators: Use cases

- Time profiling
- Maintaining our own local scope
- Simpler alternative to some of the use cases of metaclasses. For example: Singleton class

Context Managers: With statement

- Helps you write more expressive code and makes it easier to avoid resource leaks in your programs. For example:

```
with open('hello.txt', 'w') as f:  
    f.write('hello, world!')
```

- . Translates to

```
f = open('hello.txt', 'w')  
try:  
    f.write('hello, world')  
finally:  
    f.close()
```

Context Managers: Own Objects

- What's a context manager? It's a simple "protocol" (or interface) that your object needs to follow so it can be used with the **with** statement. Basically all you need to do is add `__enter__` and `__exit__` methods to an object if you want it to function as a context manager.

Context Manager: Own Objects Example

```
class ManagedFile:
    def __init__(self, name):
        self.name = name

    def __enter__(self):
        self.file = open(self.name, 'w')
        return self.file

    def __exit__(self, exc_type, exc_val, exc_tb):
        if self.file:
            self.file.close()
```


Context Manager: Own Objects Example Continue

- Can be used as follows:

```
>>> with ManagedFile('hello.txt') as f:  
...     f.write('hello, world!')  
...     f.write('bye now')
```

Context Managers: Use Cases

- Used to manage the safe acquisition and release of system resources
- For example: file lock, socket connection, etc

Comprehensions: List

- Syntax:
 - `new_list = [expression for member in iterable]`
- **expression** is the member itself, a call to a method, or any other valid expression that returns a value. In the example above, the expression `i * i` is the square of the member value.
- **member** is the object or value in the list or iterable. In the example above, the member value is `i`.
- **iterable** is a list, [set](#), sequence, [generator](#), or any other object that can return its elements one at a time. In the example above, the iterable is `range(10)`.

Comprehension: List Continue

- Syntax:

```
new_list = [expression for member in iterable (if conditional)]
```

- Example:

```
>>> sentence = 'the rocket came back from mars'
>>> vowels = [i for i in sentence if i in 'aeiou']
>>> vowels
['e', 'o', 'e', 'a', 'e', 'a', 'o', 'a']
```

Comprehensions: List + Walrus

- The formula expression `for` member `in` iterable (`if` conditional)
- provides no way for the conditional to assign data to a variable that the expression can access.

```
>>> import random
>>> def get_weather_data():
...     return random.randrange(90, 110)
>>> hot_temps = [temp for _ in range(20) if (temp :=
get_weather_data()) >= 100]
>>> hot_temps
[107, 102, 109, 104, 107, 109, 108, 101, 104]
```

Comprehensions: List Nested

- Example:

```
>>> cities = ['Austin', 'Tacoma', 'Topeka', 'Sacramento', 'Charlotte']
>>> temps = {city: [0 for _ in range(7)] for city in cities}
>>> temps
{
    'Austin': [0, 0, 0, 0, 0, 0, 0],
    'Tacoma': [0, 0, 0, 0, 0, 0, 0],
    'Topeka': [0, 0, 0, 0, 0, 0, 0],
    'Sacramento': [0, 0, 0, 0, 0, 0, 0],
    'Charlotte': [0, 0, 0, 0, 0, 0, 0]
}
```

Comprehensions: Generators

- If you were to sum the first billion squares with a generator, then your program will likely run for a while, but it shouldn't cause your computer to freeze. The example below uses a generator:

```
>>> sum(i * i for i in range(1000000000))  
333333332833333333500000000
```

Closures: What it is

- A closure simply causes the inner function to remember the state of its environment when called. Beginners often think that a closure is the inner function, but it's really caused by the inner function. The closure “closes” the local variable on the stack, and this stays around after the stack creation has finished executing.

Closures: Example

```
def generate_power(number):
```

```
    """
```

```
    Examples of use:
```

```
>>> raise_two = generate_power(2)
```

```
>>> print(raise_two(7))
```

```
128
```

```
    """
```

```
# Define the inner function ...
```

```
def nth_power(power):
```

```
    return number ** power
```

```
# ... that is returned by the factory function.
```

```
    return nth_power
```

References

- <https://realpython.com/primer-on-python-decorators>
- <https://github.com/spulec/freezegun>
- <https://realpython.com/inner-functions-what-are-they-good-for>
- <https://dbader.org/blog/python-context-managers-and-with-statement>
- <https://realpython.com/list-comprehension-python/>

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day 15

Hands on



Decorators & Context Managers - Handson

1. Make a multiplier closure function which returns back a function that can be passed any number to multiply by the factor given before.

For example:

```
make_double = multiply(2)
make_double(5) # should print "10"
```

1. Log function arguments and return values using the decorator.

For example:

```
>>> make_greeting("Benjamin")
Calling make_greeting('Benjamin')
'make_greeting' returned 'Howdy Benjamin!'
'Howdy Benjamin!'
```

```
>>> make_greeting("Richard", age=112)
Calling make_greeting('Richard', age=112)
'make_greeting' returned 'Whoa Richard! 112 already, you are growing up!'
'Whoa Richard! 112 already, you are growing up!'
```

```
>>> make_greeting(name="Dorrisile", age=116)
Calling make_greeting(name='Dorrisile', age=116)
'make_greeting' returned 'Whoa Dorrisile! 116 already, you are growing up!'
'Whoa Dorrisile! 116 already, you are growing up!'
```

Decorators & Context Managers - Handson

3. Calculate time taken by function in the function execution using decorator.

For example:

```
>>> waste_some_time(1)
```

Finished 'waste_some_time' in 0.0010 secs

```
>>> waste_some_time(999)
```

Finished 'waste_some_time' in 0.3260 secs

Decorators & Context Managers - Handson

4. Create a decorator which makes a class singleton.

In software engineering, the singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system.

Reference: https://en.wikipedia.org/wiki/Singleton_pattern

When used on class It should show similar output as "TheOne" class as follows:

```
>>> first_one = TheOne()
>>> another_one = TheOne()
```

```
>>> id(first_one)
140094218762280
```

```
>>> id(another_one)
140094218762280
```

```
>>> first_one is another_one
True
```

Decorators & Context Managers - Handson

5. File locking is a mechanism that restricts access to a computer file, or to a region of a file, by allowing only one user or process to modify or delete it in a specific time and to prevent reading of the file while it's being modified or deleted. It can be used in scenarios where an API only allows one access_key generation. This creates race condition when multiple processes try to generate access_key and only one process which finishes last has the valid access_key and all other fails the subsequent calls to the API.

The problem statement is: Implement file lock using context manager.

The file lock class should allow following syntax and when ran simultaneously on two processes, it should raise error.

```
>>> with file_lock:  
...     # do something
```

6. Write decorator function to implement caching mechanism in the fibonacci function without modifying the existing fibonacci function.

Day - 16 Code Design



Topics to be covered

- Coding Best practices
- SOLID principles
- Code Examples
- Clean coding

Code Design

- Why do we follow **Best Practices**, Documentation, Clean coding?
- Understanding the requirements first
 - Code should be expandable in future vs. Over Engineering.
- Code design
 - Clean coding (python related and in general)
 - Designing a function (modularity/abstraction)
 - Private and Public methods
 - Parameters: class obj/func obj
 - Reducing indentation
 - loosely coupled
 - Look for python best practices - **Simple is Best!**
 - Consistency in design.. V. V. important.
 - Reducing the LOC

SOLID Principles:

Why do you need to know?

- Easy to understand the codebase
- Easy to extend
- Easy to maintain the codebase
- Robust code
- Minimum changing existing codebase or not at all.

Let's deep dive into SOLID principles. SOLID is a short form. It stands for

1. Single Responsibility Principle
2. Open and Closed Principle
3. Liskov Substitution Principle
4. Interface Segregation Principle
5. Dependency Inversion Principle

Code Examples

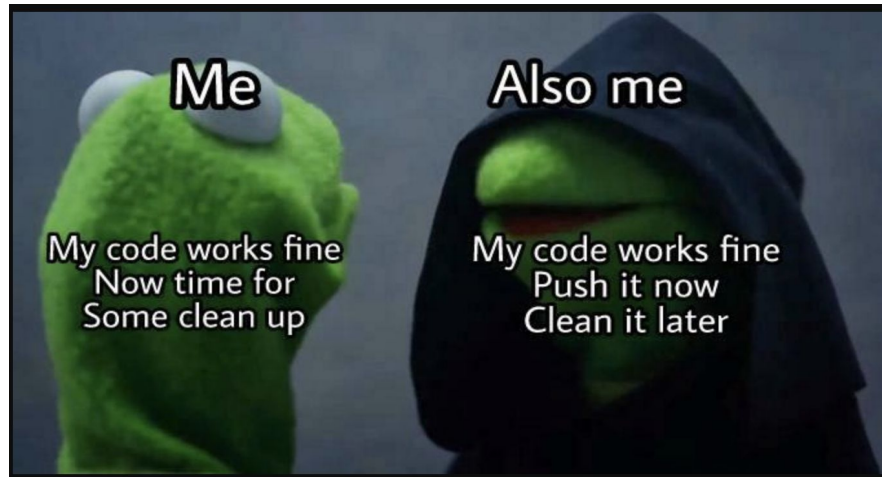
- Excellent example: Square subclasses Rectangle. This may seem acceptable at first but the difficulties become apparent when we consider the behaviour of each class when manipulating their Height and Width properties.
 - <https://github.com/heykarimoff/solid.python/blob/master/3.lsp.py>
- This principle aims to ensure that clients are not forced to depend on methods which they do not use. A good way of ensuring this is by separation through multiple inheritance. In Python we are free to inherit from multiple concrete classes, and this is precisely the purpose of the mix-ins discussed above – to provide multiple clients specific behaviours.
 - <https://github.com/heykarimoff/solid.python/blob/master/4.isp.py>
- A good example of this is a method which checks an object's length before looping over it: in the future it may be passed an object (such as a generator which is fine to iterate over but doesn't have a length. By checking the length of the object we have created rigidity in the design. It may be this is absolutely necessary but it should be a conscious decision in the design of the method.
 - <https://github.com/heykarimoff/solid.python/blob/master/5.dip.py>

Clean Coding

What is clean code?

"I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well."

- Bjarne Stroustrup, inventor of the C++ programming language



Clean Coding

Writing clean code comes with a lot of advantages: improving your software quality, code maintainability, and eliminating technical debt.

Technical debt is any code that you've decided is a liability.

Technical debt you want to avoid:

- Knowledge-based tech debt
- Design debt
- Code debt

Patterns for writing clean code in Python

Naming convention

- Use long descriptive names that are easy to read
- Use descriptive intention revealing names
- Avoid using ambiguous shorthand
- Always use the same vocabulary
- Don't use magic numbers

Functions

- Be consistent with your function naming convention
- Functions should do one thing and do it well
- Do not use flags or Boolean flags

Classes

- Do not add redundant context

References:

- https://www.researchgate.net/publication/323935872_SOLID_Python_SOLID_principles_applied_to_a_dynamic_programming_language
- <https://github.com/heykarimoff/solid.python>
- <https://medium.com/@shivama205/dependency-injection-python-cb2b5f336dce>
- <https://refactoring.guru/design-patterns/python>
- <https://medium.com/@andreaspyias/design-patterns-a-quick-guide-to-facade-pattern-16e3d2f1bfb6>
- <https://dev.to/alexomeyer/10-must-know-patterns-for-writing-clean-code-with-python-56bf>

Day - 16 Hands on



Handson

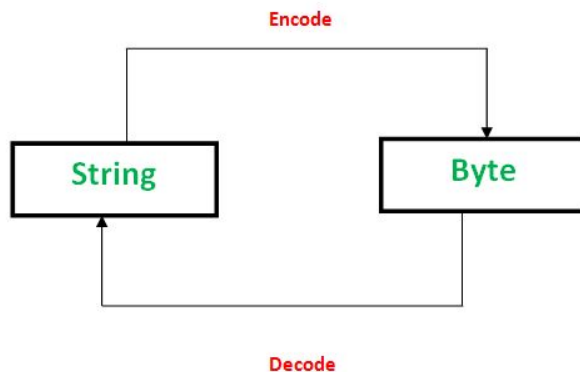
- Design classes, their objects and methods for the game **Among Us**.

Day - 17 Unicode and characters



Byte Objects vs String

- Byte objects are sequence of Bytes, whereas Strings are sequence of characters.
- Byte objects are in machine readable form internally, Strings are only in human readable form.
- Since Byte objects are machine readable, they can be directly stored on the disk. Whereas, Strings need encoding before which they can be stored on disk.
- PNG, JPEG, MP3, WAV, ASCII, UTF-8 etc are different forms of encodings. An encoding is a format to represent audio, images, text, etc in bytes. Converting Strings to byte objects is termed as encoding. This is necessary so that the text can be stored on disk using mapping using ASCII or UTF-8 encoding techniques.



ASCII

- It is acronym for the American Standard Code for Information Interchange
- It is a code for representing 128 english characters as numbers, with each letter assigned a number from 0 to 127. For example ASCII code for uppercase M is 77.
- ASCII table: <https://www.lookuptables.com/text/ascii-table>
- A standard ASCII character set uses just 7 bits for each character. There are several larger characters uses 8 bits, which gives them 128 additional characters. The extra characters are used to represent non-english characters. Symbols and mathematical symbols
- Extended ASCII table: <https://www.lookuptables.com/text/extended-ascii-table>
- How to store characters of different languages? Like ગુજરાતી

Unicode

- An international encoding standard for use with different languages and scripts, byt which each letter, digit, or symbol is assigned a unique numeric values that applies across different platforms and programs.
- Unicode is capable of encoding about at least 1,114,112 characters! It covers current and historical scripts, alphabets, symbols, emojis and non-printable codes for controlling and formatting.
- Unicode Characters and Blocks: <https://www.lookuptables.com/text/unicode-characters>
- 🐍 - Snake Emoji: U+1F40D

Encoding in python

- In python, The encode() method encodes the string, using the specified encoding. If no encoding is specified, UTF-8 will be used.

```
string.encode(encoding=encoding, errors=errors)
```

```
str = 'CrestDataSystems'  
encoded_str = str.encode('ASCII')  
print(encoded_str)  
>> b'CrestDataSystems'
```

```
Str = 'À'  
encoded_str = str.encode('ASCII')  
>> UnicodeEncodeError: 'ascii' codec can't encode character '\u0a8f' in position  
0: ordinal not in range(128)
```


Unicode Transformation Format (UTF)

- An algorithmic mapping from every Unicode code point to a unique byte sequence
- Most texts in documents and webpages is encoded using some of various UTF encodings
- UTF-8
 - The most popular type of Unicode encoding
 - 8 means that 8-bit values are used in the encoding
 - It uses only byte for standard English letters and symbols, two bytes for additional Latin and Middle Eastern characters, and three bytes for Asian characters
 - Any additional characters can be represented using four bytes
 - UTF-8 is backward compatible with ASCII, since the first 128 characters are mapped to the same values
- Similarly, UTF-16 and UTF-32 uses 16 and 32 bits for encoding characters

References

- <https://www.geeksforgeeks.org/byte-objects-vs-string-python/>
- <https://realpython.com/python-encodings-guide/>
- <https://towardsdatascience.com/processing-text-with-unicode-in-python-eacc226886cb>
- https://www.w3schools.com/python/ref_string_encode.asp
- https://www.tutorialspoint.com/python/string_decode.htm

UTF-8, UTF-16, UTF-32

- <https://www.youtube.com/watch?v=uTJoJtNYcaQ>

Handson

- Make a python script that takes a string as an input, convert characters of a string to bold characters(Unicode bold characters)
 - If the input is Crest Data Systems, then the output should be **Crest Data Systems**

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day - 18

Concurrent and Parallel Execution



Topics to be covered

- What is concurrency and parallelism
- CPU-Bound VS I/O-Bound
- Threading v/s Multiprocessing v/s asyncio

Concurrent & Parallel execution

Concurrency and Parallelism

What concurrency is?

- Simply put, it's doing multiple things at the same time
- Concurrency involves allowing multiple jobs to take turns accessing the same shared resources, like disk, network, or a single CPU core.
- Concurrency is then often understood as “managing” multiple jobs simultaneously. In actuality, those jobs don't really execute all at the same time. They cleverly alternate.

What parallelism is?

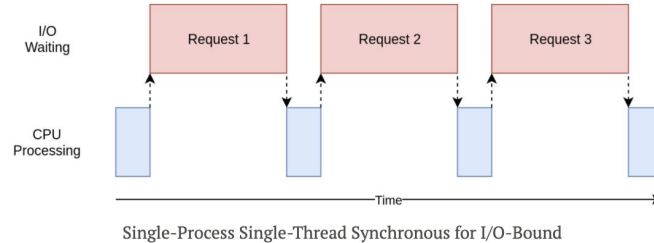
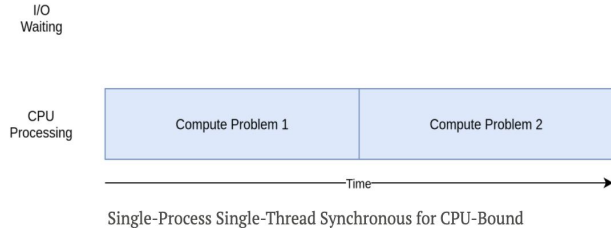
- The parallel execution, however, does mean executing multiple jobs simultaneously, or in parallel. The parallelism allows to leverage multiple cores on a single machine

In Python, concurrency is represented by threading and asyncio, whereas parallelism is achieved with multiprocessing.

Concurrent & Parallel execution

CPU-Bound VS I/O-Bound

- CPU-bound refers to a condition when the time for it to complete the task is determined principally by the speed of the central processor
- I/O bound refers to a condition when the time it takes to complete a computation is determined principally by the period spent



Example:- A common example is completing multiple network requests. Worst way to do is to launch one request, wait for it to finish, launch another, and so on. The concurrent way to do it is to launch all requests at once, then switch among them as they get responses back.

And while Parallelism, by contrast, is about maximizing the use of hardware resources. If you have eight CPU cores, you don't want to max out only one while the other seven lie idle

Concurrent & Parallel execution

Threading v/s Multiprocessing v/s asyncio

Concurrency Type	Features	Use Criteria	Metaphor
Multiprocessing	Multiple processes, high CPU utilization.	CPU-bound	We have ten kitchens, ten chefs, ten dishes to cook.
Threading	Single process, multiple threads, pre-emptive multitasking, OS decides task switching.	Fast I/O-bound	We have one kitchen, ten chefs, ten dishes to cook. The kitchen is crowded when the ten chefs are present together.
AsyncIO	Single process, single thread, cooperative multitasking, tasks cooperatively decide switching.	Slow I/O-bound	We have one kitchen, one chef, ten dishes to cook.

Summary

- What is a difference between Thread and Process?
- The way the threads or tasks take turns is the big difference between threading and asyncio. In threading, the operating system actually knows about each thread and can interrupt it at any time to start running a different thread. This is called pre-emptive multitasking since the operating system can pre-empt your thread to make the switch
- Asyncio, on the other hand, uses cooperative multitasking. The tasks must cooperate by announcing when they are ready to be switched out. That means that the code in the task has to change slightly to make this happen

References

- <https://leimao.github.io/blog/Python-Concurrency-High-Level/>
- <https://realpython.com/python-concurrency/>
- <https://realpython.com/async-io-python/>
- <https://docs.python.org/3/library/asyncio.html>
- <https://docs.python.org/3/library/multiprocessing.html>
- <https://www.analyticsvidhya.com/blog/2021/04/a-beginners-guide-to-multi-processing-in-python/>
- <https://www.velotio.com/engineering-blog/async-features-in-python>
- https://jedyang.com/post/multithreading-in-python-pytorch-using-c++-extension/featured_hu96bd47bf9e68d64336069acfc508a573_1776295_720x0_resize_lanczos_2.png

Handson

1. Download any n number of images from the site (You can use any image site for example:- <https://www.gunnerkrigg.com/comics/00000001.jpg>)
 - a. Write Synchronous version with requests lib
 - b. Write multithreading version (with Threading and Threadpoolexecutor both)
 - c. Write a async version for same
 - d. Write multiprocessing version same
 - e. Create time difference metric of all above versions and compare. Come up with the reason of multiprocessing performance.

Note:- Rather using requests.get() directly for request try to use requests.session() so multiple call use same session and check time improvement.

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day - 19 Memory leaks



Topics to be covered

- Memory leaks basics
- Memory leaks in Python
- Causes because of Memory leaks
- Factors for Memory leaks
- Identify and fix the Memory leaks

What is memory leak?

- A memory leak is the incorrect management of memory allocations by a computer program where the unneeded memory isn't released.
- When unused objects pile up in the memory, your program faces a memory leak

What causes memory leaks in Python?

- Memory leaks in Python happen if the garbage collector doesn't clean and eliminate the unreferenced or unused data from Python.
- Python developers have tried to address memory leaks through the addition of features that free unused memory automatically.
- However, some unreferenced objects may pass through the garbage collector unharmed, resulting in memory leaks.

Factors that may cause memory leaks

- Large objects lingering in the memory that aren't released
 - If you delete an object from the active directory service when the domain controller is offline, the object stays in the domain controller as a lingering object. It's those lingering objects that consume space leading to the occurrence of memory leaks.
- Reference styles in the code
 - A reference has an address and class information concerning objects being referenced. Assigning references doesn't create distinct duplicate objects. But, if an object is no longer in use and can't be garbage collected because it's being referenced in another place within the application, it results in memory leaks.
- Underlying libraries
 - Python uses multiple libraries for visualization, modeling, and data processing. Though Python libraries make Python data tasks much easier, they have been linked to memory leaks.

Methods to fix memory leaks

1. The use of debugging method to solve memory leaks
 - a. Debugging allows you to see where much of the Python storage memory is being applied. Then, you can go ahead and filter everything based on usage.
 - b. In case, you find objects that aren't in use, and maybe they are referenced, you can get rid of them by deleting them to avoid memory leaks.
2. Application of tracemalloc to sort memory leak issues in Python
 - a. Tracemalloc enables you to establish the use of a specific common function that is using memory within your program. It provides a track of memory usage by an object. You can apply that information to find out the cause of all the memory leaks. Once you get the objects leading to a memory leak, you can fix or even eliminate them.

References:

- <https://www.section.io/engineering-education/how-to-fix-memory-leaks-in-python/>
- <https://www.fugue.co/blog/diagnosing-and-fixing-memory-leaks-in-python.html>
- <https://youtu.be/F5iOjl3SSEI>

Handson

- Parse the larger CSV file (above 70 MB) and apply any aggregate function on data like Average, Maximum and Minimum. Make sure that memory leak should not be occurred.

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day - 20 Selenium

Topics to be covered

- Introduction to Selenium
- Locators
- Types of Locators
- Basic understanding of Page Object Model

Introduction to Selenium

What is Selenium?

- > **Selenium** is a open-source ***automated testing framework used to validate web applications across different browsers and platforms***. It supports multiple programming languages like Java, C#, Python etc to create Selenium Test Scripts.
- > Selenium Software is not just a single tool but a suite of software, each piece catering to different Selenium QA testing needs of an organization. Here is the list of tools:
 - Selenium Integrated Development Environment (IDE)
 - Selenium Remote Control (RC)
 - **WebDriver**
 - Selenium Grid
- > Use Selenium because,
 - To test All web-application, cross browser and cross platform
 - Capability of parallel tests
 - Easy to implement
 - Reusability and Integrations

Introduction to Selenium

Configure Webdriver & Hands On

Introduction to Selenium

```
1  import time
2  from selenium import webdriver
3  from selenium.webdriver.common.keys import Keys
4
5  driver = webdriver.Chrome(executable_path="./webdrivers/chromedriver")
6  driver.get('http://www.google.com/')
7  driver.maximize_window()
8
9  driver.implicitly_wait(5)
10 search_box = driver.find_element_by_xpath("//input[@type='text']")
11 search_box.send_keys("Ahmedabad")
12 search_box.send_keys(Keys.RETURN)
13 time.sleep(10)
14
15 driver.quit()
```

Find this example from [here](#)

Introduction to Selenium

Reference Links:

- [Introduction and basic](#)
- Download webdriver: <https://www.selenium.dev/downloads/>
- Python Selenium Documentation: [Selenium Documentation — Selenium 3.141 documentation](#)
- [Some Common Issue with Webdriver](#)
- [Advantages of Selenium](#)
- [Implicit and Example wait](#)

Hands On

1. Write a program that

- Open Chrome & Firefox browsers one after the other and perform all the given actions
- Resize the window
- maximize the window
- Hit the URL: *http://www.google.com*
- Print the title of the tab
- Open a new tab
- Hit the URL: *https://stackoverflow.com*
- Print the title of the tab
- Close the first window
- Close the session

Locators

- **Basically locator helps the program to interact with the web page.**
- Locators in Selenium come into action in the “perform designated actions in a defined test” step, after the Selenium WebDriver is initialized and loaded the webpage to be tested.
- A locator enables testers to select an HTML DOM element to act on.

The different locators in Selenium are as follows:

- CSS
- CSS class name
- HTML tag name
- Name attribute
- DOM structure or xpath
- Link text
- Partial link text

Locators

Locate Element by CSS ID

- The simplest method of locating an element.
- The CSS ID, stored in the *id* attribute of an HTML DOM element, is unique for every element in the page by design. Thus, an ID can uniquely identify an element.
- To use this feature, one needs to call the ***.find_element_by_id()*** method of the ***webdriver*** class.

```
from selenium import webdriver

driver = webdriver.Chrome('./chromedriver')
driver.get("https://www.python.org")

search_bar = driver.find_element_by_id("id-search-field")
```

Locators

Locate Element by CSS Class

- The class name is stored in the class attribute of an HTML tag.
- By design, a CSS class applies to a group of DOM elements.
- The ***.find_element_by_class_name()*** method only returns the **first** element with the matching

```
from selenium import webdriver

driver = webdriver.Chrome('./chromedriver')
driver.get("https://www.python.org")

# Returns first element with matching class
first_search_bar = driver.find_element_by_class_name("id-class-name")
```

Locators

Locate Element by Name

- In HTML5, form elements often have a name attribute associated with them.
- The *.find_element_by_name()* method only returns the **first** element with the matching class.
- If there are multiple elements of the same name, the first matched element would be returned.

```
from selenium import webdriver

driver = webdriver.Chrome('./chromedriver')
driver.get("https://www.python.org")

# Returns first element with matching class
first_search_bar = driver.find_element_by_class_name("id-class-name")
```

Locators

Locate Element by XPath

- If one has failed to locate an element with ID, class or name, one can locate it through is **XML path**.
- The ***.find_element_by_xpath()*** method is used to locate an appropriate element in document.
- Eg. To find the email input field of an HTML form,

```
email_input = driver.find_element_by_xpath("//form[input/@name='email']")
```

- This code snippet searches for the first form element of the page. Within this form, it searches for an input with the name which equals the value email, thus narrowing down to the required element.

Locators

Locate Multiple elements

- One may want to select a group of elements and then iterate through them.
- The *.find_elements()* method helps in finding **multiple** elements in the DOM structure.
- Various methods (by id, by xpath, by class etc) can be used with this method. Two of them are listed below

```
from selenium.webdriver.common.by import By

all_inputs = driver.find_elements(By.XPATH, '//form[@id='loginForm']/input')
```

```
from selenium.webdriver.common.by import By

all_elements = driver.find_elements(By.CLASS_NAME, 'my-css-class')
```

Locators

Reference Links:

- [POM Builder](#) : Google plugin makes your life easy in initial time.
- [Python Selenium Locators Documentation](#)
- [XPath & CSS selectors notes](#)

Hands On

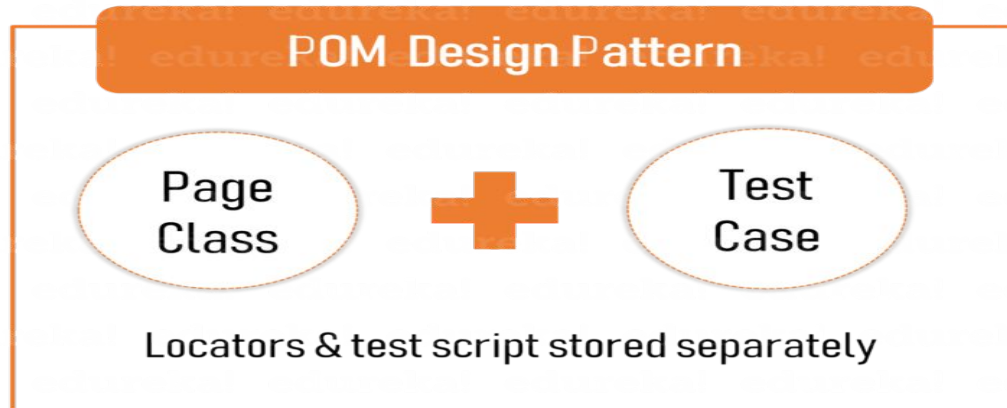
Hands-on for Locators

2. Find all possible locators from <https://demoqa.com/automation-practice-form>
3. Read the table content [from here](#) and print them in consol.
4. Login into [this page](#) (Note: you need to create a user to perform successful flow of login.)

Basic understanding of Page Object Model

What is Page object Model?

Page Object Model is a design pattern in test automation to create an Object Repository for web UI elements. Every web page in the application should have a corresponding page class. This Page class will find the WebElements and also may contain page methods which perform operations on those WebElements.



Basic understanding of Page Object Model

Why Page Object Model?

- It makes ease in maintaining the code (flow in the UI is separated from verification)
- Makes code readable (Methods get more realistic names)
- Makes the code reusable (object repository is independent of test cases)
- The Code becomes less and optimised

Basic understanding of Page Object Model - Hands on

5. Test [the Book store app](#). Perform following steps using selenium script. Follow the instructions while writing automation script:
 - Login to the system.
 - Select any two books from book store and add into collection.
 - Delete one of them.

Instructions:

- Create **DriverFactory.py** which returns the appropriate driver for firefox or chrome browser.
- Create **login.credentials** file contains the username, password required for authentication. (Optional)
- Create directory '**pages**' directory and create a base class **Page.py** under the directory. This has useful wrappers for common Selenium operations (open page, getxpath, wait)

Basic understanding of Page Object Model - Hands on (Continue)

- Create a Page model classes under the 'pages' directory like login, home_page, book_store_page, profile_page etc. These classes can have locators and common methods(click, verify, send text, get text). All page models can inherit from the Page class.
- Create **bookstore_sanity_test.py** file to run the given test cases.
- Create **helper.py** utility helper class if required.
- Strictly follow [Pylint](#) and [pycodestyle](#) linter for code quality.

References:

- [Reference video](#)
- [Page Object Models](#)
- [Config Parser \(.ini files\)](#)

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Day - 21 Celery & Packaging and Distribution



Topics to be covered

- Python Package Index (PyPI)
- Package formats on the PyPI
- Setup tools
- Build Wheel/tar of a package
- Upload to PyPI using Twine
- Introduction to Poetry: Alternative of Setuptools

Packaging and Distribution

The Python Package Index (PyPI)

- PyPI is a vast repository of open-source Python packages supplied by the worldwide community of Python developers.
- Python packages are published on the Python Package Index (<https://pypi.org>)
- `pip install package_name` will by default:
 1. Search for a package name on PyPI
 2. Download the best available distribution for your platform
 3. Install all dependencies of the package
 4. Install the package

Package formats on the PyPI

- Source Distribution (tar)
 - .zip or .tar.gz archives of the project
 - Simplest solution to publish your package
- Wheels
 - Wheel is a built-package format, and offers the advantage of not re-compiling your software during every install.
 - A wheel is a ZIP-format archive with a specially formatted file name and the .whl extension
 - Platform dependent

Build Wheel/tar of a package

- Install the build package `python -m pip install build`
- Run the build module in the project directory
- Above command will generate the tar and wheel format of the package inside the sdist directory

Upload to PyPI using [Twine](#)

- Twine is the primary tool developers use to upload packages to the Python Package Index or other Python package indexes.
- Install twine using `python -m pip install twine`
- Create the account on the PyPI site
- Run the upload command of twine `twine upload dist/*`

Introduction to Poetry: Alternative of Setuptools

- Poetry is a tool for dependency management and packaging in Python.
- It allows you to declare the libraries your project depends on and it will manage (install/update) them for you.
- Configured completely in pyproject.toml
- Automatic creation of virtual environments with all dependencies
- Exact versions of all dependencies (including transitive) using a “lock file” + Building and publishing packages
- Initial setup of new projects
- Poetry project for reference: <https://github.com/hackandslackers/python-poetry-tutorial>

Using Setuptools

- Setuptools is the most common solution for python packaging
- Setuptools is a package development process library designed to facilitate packaging Python projects by enhancing the Python standard library distutils (distribution utilities).
- Used to declare package metadata, dependencies
- Package structure: <https://github.com/pypa/sampleproject>

Setup files: setup.py and setup.cfg

- All metadata concerning package can be specified in a setup.py/setup.cfg file
- Setup.py: <https://github.com/pypa/sampleproject/blob/main/setup.py>
- Example

setup.py

```
from setuptools import setup

setup( name="foobar",
       version="0.1.0",
       author="John Doe",
       # ... )
```

setup.cfg

```
[metadata]
name = foobar
version = 0.1.0
author = John Doe
# ...
```

Package Structure

```
.
├── LICENSE.txt
├── README.md
├── crest_demo
│   ├── __init__.py
│   ├── crest_demo.py
│   └── tests
│       ├── __init__.py
│       └── test_crest_demo.py
├── setup.cfg
└── setup.py
```

References:

- <https://pypi.org/>
- <https://packaging.python.org>
- <https://python-poetry.org/docs/>

Handson

- Build a small library, package it using setuptools, and upload it to Test PyPI. Configure maximum possible parameters of setup method in setup tools.

What is Celery?

- Celery is a simple, flexible, and reliable distributed system to process vast amounts of messages, while providing operations with the tools required to maintain such a system.
- It's a task queue with focus on real-time processing, while also supporting task scheduling.

What do we need?

- Celery requires a message transport to send and receive messages. The RabbitMQ and Redis broker transports are feature complete, but there's also support for a myriad of other experimental solutions, including using SQLite for local development.
- Celery can run on a single machine, on multiple machines, or even across data centers.

What is broker?

- Medium for message transfer

What is Backend?

- Database

Steps:

- `python3 -m venv venv`
- `pip install celery`
- Install broker and backend
 - `brew install rabbitmq`
 - `export PATH=$PATH:/usr/local/sbin`
 - `rabbitmq-server`
 - Ref : <https://www.rabbitmq.com/install-homebrew.html>
 - Windows
 - <https://www.rabbitmq.com/install-windows.html#installer>
- To run the celery App
 - `celery -A proj worker -l info`
 - `celery -A proj flower --address=127.0.0.1 --port=5599`
 - `celery -A proj beat`

Sample Celery Application

References:

Celery :

- <https://www.rabbitmq.com/install-windows.html#installer>
- <https://pypi.org/project/SQLAlchemy/>
- <https://docs.celeryproject.org/en/stable/getting-started/first-steps-with-celery.html>
- <https://docs.celeryproject.org/en/stable/getting-started/backends-and-brokers/index.html>
- <https://docs.celeryproject.org/en/stable/getting-started/backends-and-brokers/rabbitmq.html#broker-rabbitmq>
- <https://docs.celeryproject.org/en/stable/getting-started/backends-and-brokers/redis.html#broker-redis>
- <https://www.youtube.com/watch?v=THxCy-6EnQM>
- <https://www.fullstackpython.com/celery.html>
- <https://www.youtube.com/watch?v=fBfzE0yk97k>
- <https://www.youtube.com/watch?v=7dUxylp5mMI>
- <https://www.youtube.com/watch?v=kWxYPq7Sc8A>
- <https://www.youtube.com/watch?v=jeE-k5dG8ug>
- <https://drive.google.com/file/d/1e4iVwh8qIDHW3F2nv-Zm2XL-OGmztrS6/view?usp=sharing>

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

Handson

Celery :

- Design Celery app that fetches data at periodic interval from third party API and store it in sqlite3 database

Packaging & distribution:

- Build a small library, package it using setuptools and poetry, and upload it to PyPI

Topics to be covered

- Classic Service
- Monolithic applications
- Microservice
- Advantages and Disadvantages of using Microservice Architectures

Day - 22

Microservice

Topics to be covered

- What is a service?
- What is a microservice?
- What are monolithic applications?
- Adv and Disadv of using microservice architectures?

Microservice Architecture (Contd..)

What are monolithic applications?

- A monolithic architecture is a traditional model.
- A monolithic architecture is a singular, large computing network with one code base that couples all of the business concerns together.
- To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming.
- Adv: Easy deployment, development, performance, simplified testing, easy debugging
- Dis-adv: Slower development speed, scalability, reliability, barrier to technology adoption, lack of flexibility, deployment
- Ref links: [link](#)

Microservice Architecture

What is a service?

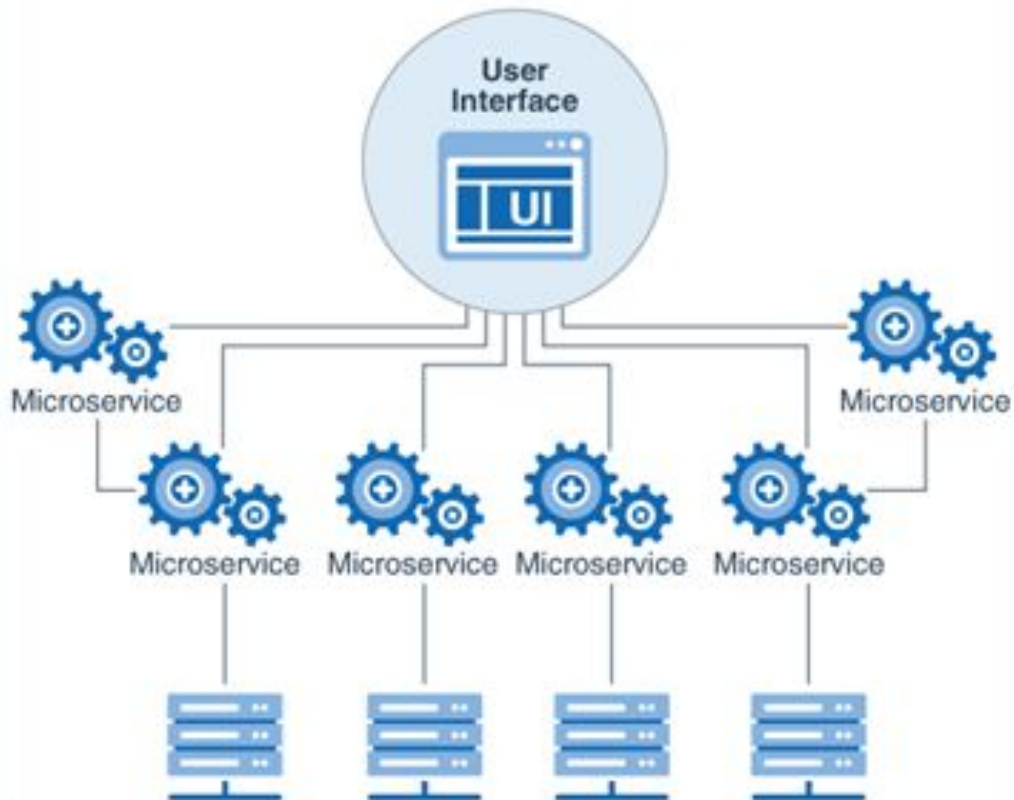
- Services are essentially groups of software components that help a company seamlessly carry out important business processes
- A service is commonly characterized by these four properties:
 1. It logically represents a business activity with a specified outcome.
 2. It's self-contained.
 3. It's a black box for its consumers.
 4. It may consist of other underlying services.
- Ref link:
<https://www.innovativearchitects.com/KnowledgeCenter/business-connectivity/SOADefinitionAndCharacteristics.aspx>

Microservice Architecture (Contd..)

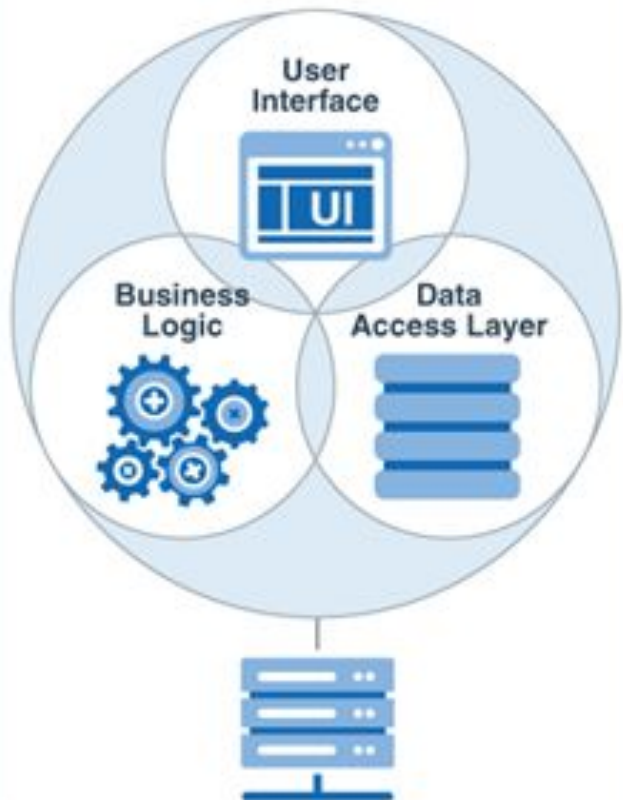
What is a microservice?

- A microservice architecture – a variant of the service-oriented architecture structural style – is an architectural pattern that arranges an application as a collection of loosely-coupled, fine-grained services, communicating through lightweight protocols.
- One of its goals is that teams can develop and deploy their services independently of others.
- It is a development method that breaks down software into modules with specialized functions and detailed interfaces
- “Do one thing and do it well”
- Ref links:
 - <https://en.wikipedia.org/wiki/Microservices>
 - <https://microservices.io/>

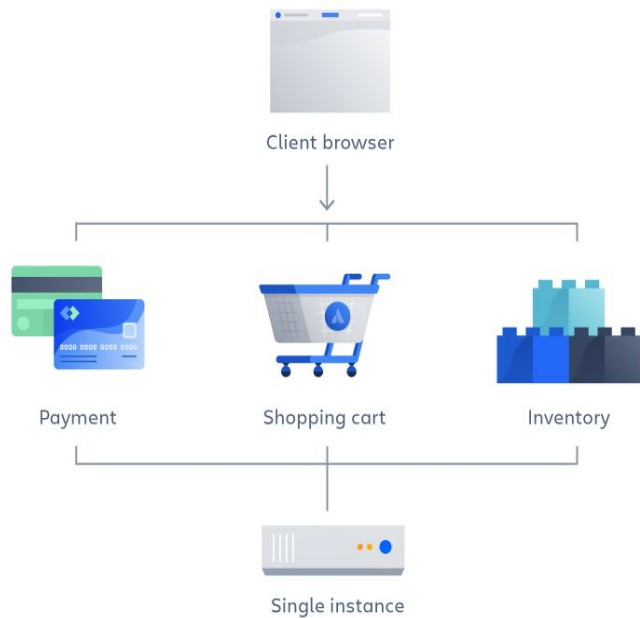
Microservice Architecture



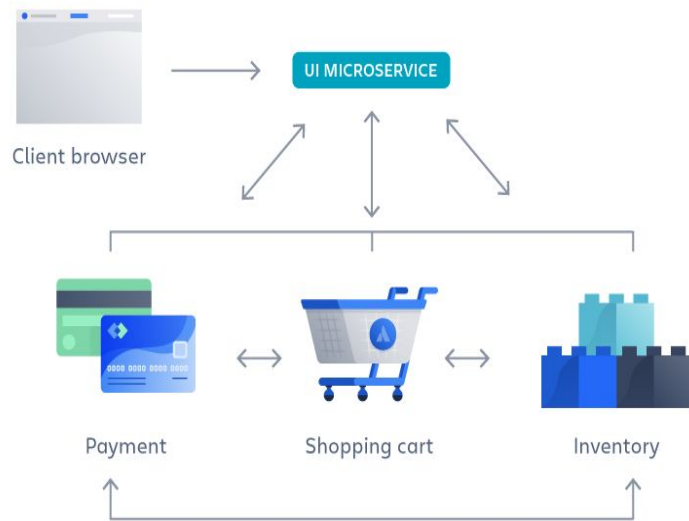
Monolithic Architecture



Monolithic architecture



Microservice architecture



Microservice Architecture (Contd..)

Pros:

- Agility
- Flexible scaling
- Continuous deployment
- Highly maintainable and testable
- Independently deployable
- Technology Flexibility
- High Reliability
- Ref links: [Link](#)

Microservice Architecture (Contd..)

Cons:

- Development Sprawl
- Exponential infrastructure costs
- Added organizational overhead
- Debugging challenges
- Lack of standardization
- Ref links: [Link](#)

Handson

Create 2 microservices:

1. This would expose an API layer to end user to achieve some tasks.
2. This would provide APIs to the first service and will communicate with the database

Groups of 4 with 2 teams each of 2:

Requirements:

1. Documentation of 4-8 apis of the services.
2. Use flask/fast api for your APIs.
3. Using proper modules and classes in individual services.
4. When both the services are run, the services should be able to communicate with each other.

References



- <https://en.wikipedia.org/wiki/Microservices>
- <https://microservices.io/>

Q & A session

Note: There will be only Q & A session today. No new topics will be covered.

Doubt Solving Session at : 6:30 pm to 7:00 PM

THANK YOU!

 info@crestdatasys.com
 <http://www.crestdatasys.com/>

