Meet A. Raut
S21 84

A++

14/03/24

Assignment 1

Q.1) Explain the manipulation functions of list, Tupple, dictionary and set in detail.

Ans. • Lists:

1. Append: list.append(x) adds an item x to the list at end.

2. Extending: list.extend(iterable) extends the list by appending elements from the iterable

3. Inserting: list.insert(i,x) inserts an item x at a specified index

4. Removing: list.remove(x) removes the first occerence of item x.

5. Popping: list.pop([i]) removes and returns the item at index i. If no index is specified, it removes and returns the last item.

6. Clearing: list.clear() removes all items from the list

7. Counting: list.count(x) returns the number of occerences of item x.

8. Sorting: list.sort() sorts the items of the lists in place.

• Tuples:

Tupes are the same as lists with the exception that the data once entered cannot be changed no matter what.

User can create, access and append to the tupple.

- ~~Tuples~~ Dictionaries:

1) Accessing: dict[key] retrieves the value associated with the specified key

2) Adding : dict[Key] = value adds a new key-value pair

3) Removing: ~~dict~~ del dict[key] removes the item with the specified key

4) Getting keys: dict.keys() returns a view of all keys in the dictionary.

5) Clearing: dict.clear() removes all items from the dictionary

- Sets:

1) Adding: set.add(x) adds an element x to the set.

2) Clearing: set.clear() removes all elements from the set

3) Updating: set.update(iterable) updates the set by adding elements from the iterable.

4) Discarding: set.discard(x) removes element x from the set if it is present

5) Operations: Union, intersection, difference, Symmetric difference, subset, ~~&~~ superset, disjointness are also common set operations

Q. Explain the constructs (polymorphism, encapsulation, inheritance) of OOP in detail.

Ans.

§* Encapsulation:

§ Encapsulation is the bunding of data and methods that operates on that data into a ~~single~~ single unit, called a class. In python, this is achieved through the use of classes. A class serves as a blueprint for creating objects, and it encapsulates both data attributes and methods that operate on those attributes. By encapsulating data and methods within a class we can controlf access to the data, making it possible to enforce constraints and ensure data integrity.

```
Class Car :
    def --init--(self, make, model):
        self.make = make          #public
        self._model = model       # protected
        self.--year = 2022        # private

    def get-year (self):
        return self.--year

car = Car("Toyota", "Camry")
print (car.make)      # public
print(car._model)     # protected
print (car. get-year ()) # Accessing private attribute through method.
```

* Inheritance :

Inheritance is a mechanism in OOP that allows a new class to inherit properties and behavior from an existing class. In python, inheritance is achieved by specifying the superclass in paranthesses after the subclass ~~n~~ name ~~s~~ when defining the subclass.

The subclass inherits all attributes and methods of its superclass and can also define its own additional attributes and methods.

```
class Vehicle :
    def __init__(self, make, model) :
        self.make = make
        self.model = model

    def display_info(self) :
        return f"{self.make} {self.model}"

Class Car(Vehicle):
    def __init__(self, make, model, year):
        super().__init__(make, model)
        self.year = year

    def display_info(self):
        return f"{~~sel~~ self.year} {self.make} {self.model}"

car = car("Toyota", "camry", 2022)
print(car.display_info())
```

* Polymorphism:

polymorphism is the ability of different objects to respond to the same message in different ways. In python, polymorphism is achieved through method ~~overloading~~ overriding and method overloading.

~~Method overloading all~~

Method overring allows a subclass to provide a specific implementation of a method that is already defined in its superclass.

Method overloading, on the other hand is not directly supported in Python as it is in some other languages like JAVA, but ~~y~~ we can achieve a similar effect using default parameter values or variable length argument lists.

```
class Animal:
    def speak(self):
        pass

Class Dog(Animal):
    def speak(self):
        return "Woof!"

def make-sound(animal):
    return animal.speak()

dog = Dog()

print(make_sound(dog))
```

Q.3) Explain different anonymous function with examples.

Ans.

Anonymous functions, also known as lambda functions, are small inline functions that can be defined without a proper name or a def keyword. They are typically used for short, one-time operations where defining a full function using def is unnecessary.

Here are some different types of annonymous functions with examples:

1. Basic lambda function:

It takes one or more arguments and returns a value.

```
add = lambda x, y: x+y
print(add(3, 5))
```

2. Lambda function with conditional expression:

A lambda function with a conditional expression.

```
check_even = lambda x: True if x % 2 == 0 else False
print(check_even(4))
```

3. Lambda function as a key function:

Using labda function as a key ~~for~~ function for sorting.

```
names = ['Alice', 'Bob', 'Charlie', 'David']
sorted_names = sorted(names, key = lambda x: len(x))
print(sorted_names)
```

4. Lambda function with default arguments:

It is a function with default arguments.

```
power = lambda x, y=2: x ** y
print (power(3))
print ( power(3, 3))
```

5. Lambda function for mapping:

Using lambda functions with map() to apply a function to every element in an iterable.

```
numbers = [1, 2, 3, 4, 5]
squared-numbers = list (map( lambda x : x ** 2, numbers))
print(squared-numbers)
```

6. Lambda function for filtering:

Using lambda functions with filter () to filter elements from an iterable.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = list(filter(lambda x: x % 2 ==0, numbers))
print (even_numbers)
```

Lambda functions provide a concise way to define small function without the need for ~~an~~ a formal function defination. They are especially useful in scenarios where a simple operations needs to be performed inline, such as in sorting, filtering or mapping operation.

Meet Raut
S21 84

A11

Gaurab
26/03/24

Assignment 2

Q.1)

Ans. 1. Reading and writing data :

a) pd.read_csv():

Reads data from a CSV file into a DataFrame

b) pd. read-excel ():

Reads data from an Excel File into a Dataframe

c) pd. to_csv() :

Writes dataframe into a CSV file

d) pd. to- excel() :

Writes Dataframe to an Excel file.

2. Viewing data :

a) df. head (n) :

Returns the first n rows of the dataframe

b) df. tail (n) :

Returns the last n rows of the dataframe

3. Fitering daba :

a) df[df['column'] condition] :

Filters rows based on a condition

b) df. query ('condition') :

Filters rows using a query expression.

4. Sorting data :

a) df.sort- values (by = 'column') :

Sorts the Dataframe by the values in a column.

b) df. sort_index():

Sorts the dataframe by index

5. Statistical Functions:

a) df. describe():

Generates descriptive statistics of the Dataframe

b) df. mean()

df. median()

df. std()

:

Calculates various statistical measures.

Q.2)
Ans. Python supports various file operations, including:

1. Opening a file
2. Reading from a file
3. Writing to a file
4. Appending to a file
5. Closing a file

Code:

```
# Opening a file
file-path = "example.txt"
file = open(file_path, "r")

# Reading from a file
content = file.read()
print("File content:")
print(content)

# Closing a file
file.close()

# Writing to a file
file = open(file_path, "w")
file.write("This is new line")
file.close()
```

```
# Appending to a file
file = open (file-path, "a")
file.write ("This line is oppending"
file.close()
```

In this example:

1) We first open a file in read mode and print it's content
2) Then, we open the same file in write mode and write some content to it.
3) After that, we open file again in append mode and append some more content to it.

Q.3)

Ans. Networking in python refers to the ability to communicate and exchange data between computers over a network using Python Python offers several built-in modules like socket, http, urllib & third-party libraries like requests for handling networking tasks.

1. Socket Programming:

It provides low-level networking interface for creating client-server applications, supporting both TCP and UDP protocols It allows creating sockets, binding them to specific addresses and ports, and establishing connections for data transmission.

2. HTTP Requests:

It provides functionalities for making HTTP requests and handling responses. It allows fetching web pages, sending POST requests, handling cookies, and more.

3. URL Handling:

It provides utilities for handling URL's including parsing fetching and opening resources across the network.

4. Third-party libraries:

Libraries like requests simplify HTTP requests by providing a higher-level interface compared to the built-in module, making it easier to work with APIs and web services