**Summary Notes - Day 6**

**Name : Podutur Lahari - DE126**

**Date : 13-11-2024**

**Python Lambda Functions Overview**

* Lambda functions are anonymous (nameless) functions in Python, defined using the lambda keyword.
* Syntax: lambda arguments : expression
  + Can take any number of arguments but only one expression.
  + Useful in contexts requiring function objects with single expressions.

Basic Example

* Lambda to convert string to uppercase:

upper = lambda string: string.upper() print(upper("Hexforhexa"))

*# Output: HEXFORHEXA*

Use Cases

1. Condition Checking:
   * Format numbers with lambda, switching between scientific and comma formatting based on type.

format\_numeric = lambda num: f"{num:e}" if isinstance(num, int) else f"{num:,.2f}"

1. Def vs. Lambda Functions:
   * def supports multi-line blocks; lambda is single-line.
   * Both achieve similar results for simple calculations.

Practical Applications

* List Comprehension:
  + Generating lists with lambda for transformations.
  + Example: Multiplying list elements by 10.
* Lambda with if-else:
  + Max function to find the larger of two numbers:

Max = lambda a, b: a if a > b else b

* Multiple Lambda Calls:
  + Chain lambdas for more complex operations like sorting and finding second largest in nested lists.

Lambda with Built-in Functions

1. Filter:
   * Used to filter elements from a list based on conditions.
   * Example: Extract odd numbers.
2. final\_list = list(filter(lambda x: x % 2 != 0, li))
3. Map:
   * Applies lambda function to each list element.
   * Example: Converting list items to uppercase.
4. Reduce (from functools):
   * Reduces list to a single value by applying lambda iteratively.
   * Example: Summing all elements.

Arbitrary Arguments in Lambda

* \*args: For variable positional arguments (e.g., summing values).
* \*\*kwargs: For variable keyword arguments (e.g., storing address details in dictionary format).

Examples with Variable Arguments

* Mixed argument types (positional, keyword, arbitrary) are supported, with order restrictions for correct parsing

**CSV Basics**

- CSV (Comma Separated Values) format is widely used for storing tabular data as plain text.

- Python’s `csv` library simplifies reading and writing CSV files.

Reading CSV Files

1. Using `csv.reader` :

* Steps: Import `csv`, open file, use `csv.reader`, read header and rows, and close file.
* Example:

import csv

with open('file.csv', 'r') as file:

reader = csv.reader(file)

header = next(reader)

rows = list(reader)

2. Using `.readlines()`:

* Reads file lines into a list without `csv` library.
* Example:

with open('file.csv') as file:

lines = file.readlines()

3. Using `pandas`:

`pd.read\_csv()` reads data into a DataFrame for easier manipulation.

Example:

import pandas as pd

data = pd.read\_csv("file.csv")

4. Using `csv.DictReader`:

Reads CSV into a list of dictionaries (field names as keys).

Example:

import csv

with open('file.csv', 'r') as file:

reader = csv.DictReader(file)

for row in reader:

print(row)

**Mapping Functions**

* map() applies a function to each element in an iterable.
* Syntax: map(function, iterable)
* Example :

numbers = [1, 2, 3, 4]

doubled = list(map(lambda x: x \* 2, numbers))

print(doubled) # Output: [2, 4, 6, 8]

**String Functions**

* Python provides several string methods for text manipulation.
  + upper(): Converts all characters to uppercase.
  + replace(): Replaces part of the string with another string.
  + split(): Splits the string by a delimiter into a list.
* Example:
* text = "hello world" print(text.upper()) *# Output: "HELLO WORLD"* print(text.replace("world", "Python")) *# Output: "hello Python"* print(text.split()) *# Output: ["hello", "world"]*

**Number Functions**

* Python has built-in functions and the math module for numerical operations.
  + abs(): Returns the absolute value.
  + round(): Rounds the number to the nearest integer or specified decimal places.
  + max() and min(): Find the largest and smallest values in an iterable.
* Example:
* import math print(abs(-7)) *# Output: 7* print(round(3.14159, 2)) *# Output: 3.14* print(math.sqrt(16)) *# Output: 4.0*

**Date and Time Functions**

* The datetime module provides tools for date and time manipulation.
  + datetime.now(): Gets the current date and time.
  + strftime(): Formats date/time.
  + timedelta: Used for date arithmetic.
* Example:
* from datetime import datetime, timedelta now = datetime.now() print(now) *# Output: 2024-11-13 12:45:30 (current date & time)* print(now.strftime("%Y-%m-%d")) *# Output: 2024-11-13 (formatted)* tomorrow = now + timedelta(days=1) print(tomorrow) *# Output: 2024-11-14*

**Python Functions**

* Defined using the def keyword; they encapsulate code that can be reused.
* Example:
* def greet(name): return f"Hello, {name}!" print(greet("Alice")) *# Output: Hello, Alice!*

**Default Argument Values**

* You can assign default values to parameters, making them optional.
* Example: def greet(name="Guest"): return f"Hello, {name}!" print(greet()) *# Output: Hello, Guest!* print(greet("Bob")) *# Output: Hello, Bob!*

**Keyword Arguments**

* Allows passing arguments by parameter name, improving clarity.
* Example: def describe\_pet(name, animal\_type="dog"): print(f"{name} is a {animal\_type}.") describe\_pet(animal\_type="cat", name="Whiskers") *# Output: Whiskers is a cat.*

**Special Parameters**

* Use / to enforce positional-only parameters and \* for keyword-only.
* Example:
* def info(pos\_only, /, standard, \*, kw\_only): print(f"{pos\_only}, {standard}, {kw\_only}") info("Positional", "Standard", kw\_only="Keyword") *# Output: Positional, Standard, Keyword*

**Arbitrary Argument Lists**

* \*args for arbitrary positional arguments and \*\*kwargs for arbitrary keyword arguments.
* Example:

def sum\_numbers(\*args): return sum(args) print(sum\_numbers(1, 2, 3)) *# Output: 6* def print\_info(\*\*kwargs): for key, value in kwargs.items(): print(f"{key}: {value}") print\_info(name="Alice", age=25) *# Output: name: Alice, age: 25*

**Lambda Expressions**

* Anonymous, single-line functions using lambda.
* Example:

square = lambda x: x \* x print(square(5)) *# Output: 25* numbers = [1, 2, 3] doubled = list(map(lambda x: x \* 2, numbers)) print(doubled) *# Output: [2, 4, 6]*