**Python Lambda Functions**

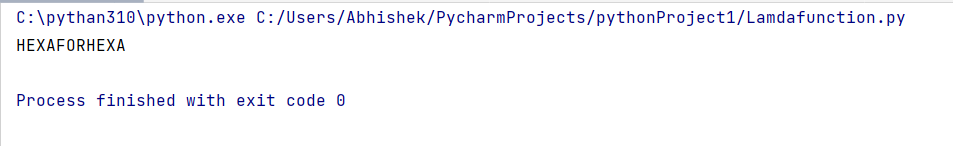
**1. Introduction to Python Lambda Functions**:

The tutorial begins by introducing Python lambda functions as anonymous functions lacking a name. The syntax and structure of lambda functions are explained, highlighting their capability to handle a single expression effectively.

**str1 = 'HexaforHexa'**

**upper = lambda string: string.upper()**

**print(upper(str1))**

****

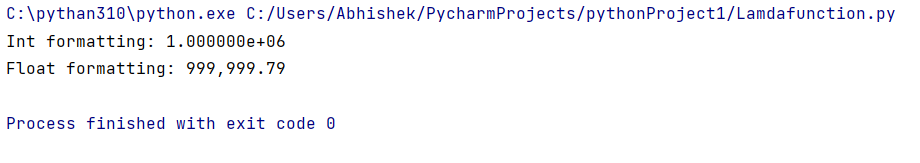
**2. Condition Checking with Lambda:**

An example illustrates the use of lambda functions for condition checking. The format\_numeric lambda function formats numeric values based on whether they are integers or floats.

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

**print("Int formatting:", format\_numeric(1000000))**

**print("Float formatting:", format\_numeric(999999.789541235))**

****

**3. Comparison with def Functions:**

A comparison is made between traditional def functions and lambda functions. The example calculates the cube of a number using both approaches.

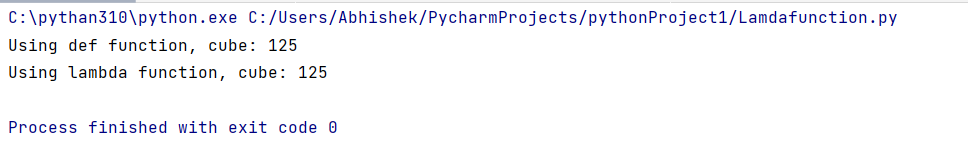
**def cube(y):**

**return y\*y\*y**

**lambda\_cube = lambda y: y\*y\*y**

**print("Using def function, cube:", cube(5))**

**print("Using lambda function, cube:", lambda\_cube(5))**

****

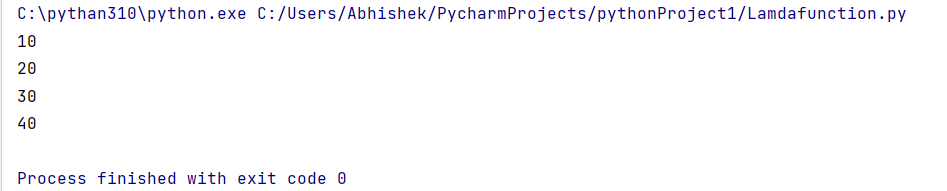
**4. Practical Uses of Lambda Functions:**

The tutorial showcases practical applications of lambda functions, including their use in list comprehension, if-else statements, and handling multiple statements.

**4.1 Lambda Functions with List Comprehension:**

**is\_even\_list = [lambda arg=x: arg \* 10 for x in range(1, 5)]**

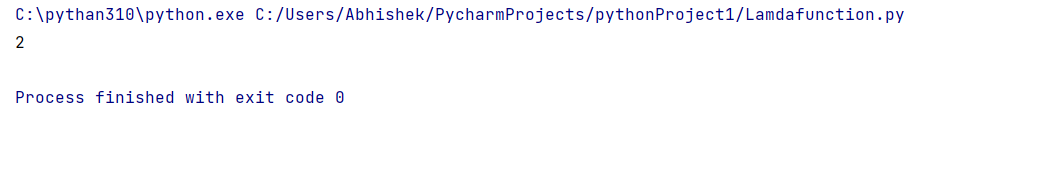
**for item in is\_even\_list:**

**print(item())  
  
**

**4.2 Lambda Functions with if-else:**

**Max = lambda a, b : a if(a > b) else b**

**print(Max(1, 2))**

****

**4.3 Lambda with Multiple Statements:**

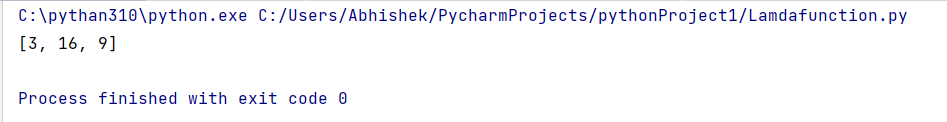
**List = [[2,3,4],[1, 4, 16, 64],[3, 6, 9, 12]]**

**sortList = lambda x: (sorted(i) for i in x)**

**secondLargest = lambda x, f : [y[len(y)-2] for y in f(x)]**

**res = secondLargest(List, sortList)**

**print(res)**

****

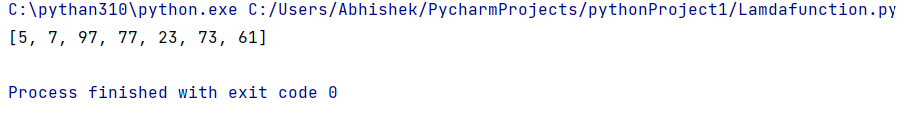
**5. Lambda Functions with Built-in Functions:**

The tutorial demonstrates using lambda functions with built-in functions such as filter(), map(), and reduce().

**5.1 Using lambda() Function with filter():**

**li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]**

**final\_list = list(filter(lambda x: (x % 2 != 0), li))**

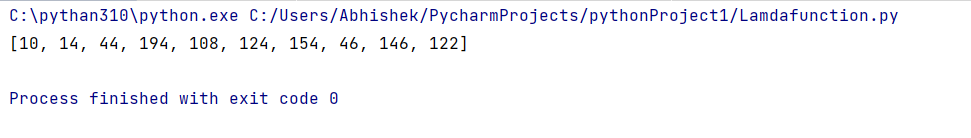
**print(final\_list)  
**

**5.2 Using lambda() Function with map():**

**li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]**

**final\_list = list(map(lambda x: x\*2, li))**

**print(final\_list)**

****

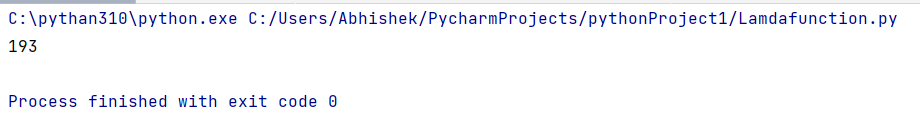
**5.3 Using lambda() Function with reduce():**

**from functools import reduce**

**li = [5, 8, 10, 20, 50, 100]**

**sum = reduce((lambda x, y: x + y), li)**

**print(sum)**

****

**6. Arbitrary Arguments and Keyword Arguments**:

The tutorial covers the use of lambda functions with arbitrary positional and keyword arguments, demonstrating their flexibility in handling variable numbers of arguments.

**def add(\*args):**

**s = 0**

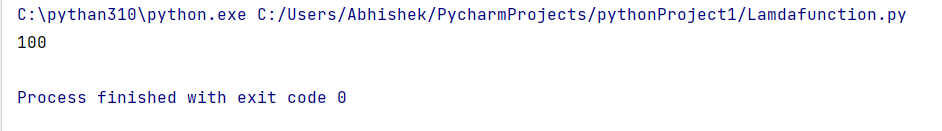
**for x in args:**

**s += x**

**return s**

**result = add(10, 20, 30, 40)**

**print(result)**

****

**7. Mixed Types of Arguments:**

An example is provided where lambda functions handle mixed types of arguments, emphasizing the importance of the order of arguments in the argument list.

**def percent(math, sci, \*\*optional):**

**s = math + sci**

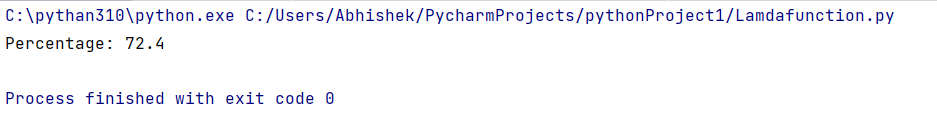
**for k, v in optional.items():**

**s += v**

**return s / (len(optional) + 2)**

**result = percent(math=80, sci=75, Eng=70, Hist=65, Geo=72)**

**print("Percentage:", result)**



**Reading JSON Strings to Python Dictionaries:**

**1.Import the json Module:**

The json module in Python provides methods for working with JSON data.

**import json**

**2.JSON String representing a Dictionary:**

Suppose you have a JSON string representing a dictionary:

**json\_dict\_string = '{"name": "John", "age": 30, "city": "New York"}'**

**3.Use json.loads():**

The json.loads() function is used to load (parse) a JSON string and convert it into a Python dictionary.

**python\_dict = json.loads(json\_dict\_string)**

**4.Accessing Values in the Dictionary:**

You can now access values in the resulting Python dictionary as you would with any other dictionary.

**print("Python Dictionary:", python\_dict)**

**print("Name:", python\_dict["name"])**

**print("Age:", python\_dict["age"])**

**print("City:", python\_dict["city"])**

**Reading JSON Strings to Python Lists:**

**1.JSON String representing a List:**

Suppose you have a JSON string representing a list:

**json\_list\_string = '[1, 2, 3, 4, 5]'**

**2.Use json.loads():**

Similar to the dictionary example, use json.loads() to parse the JSON string and convert it into a Python list.

**python\_list = json.loads(json\_list\_string)**

**3.Accessing Values in the List**:

You can access values in the resulting Python list as you would with any other list.

**print("\nPython List:", python\_list)**

**print("Third element:", python\_list[2])**

**4.Handling JSONDecodeError:**

If the JSON string is malformed or not valid JSON, the json.loads() function will raise a json.JSONDecodeError. It's a good practice to handle this potential error to ensure the robustness of your code.

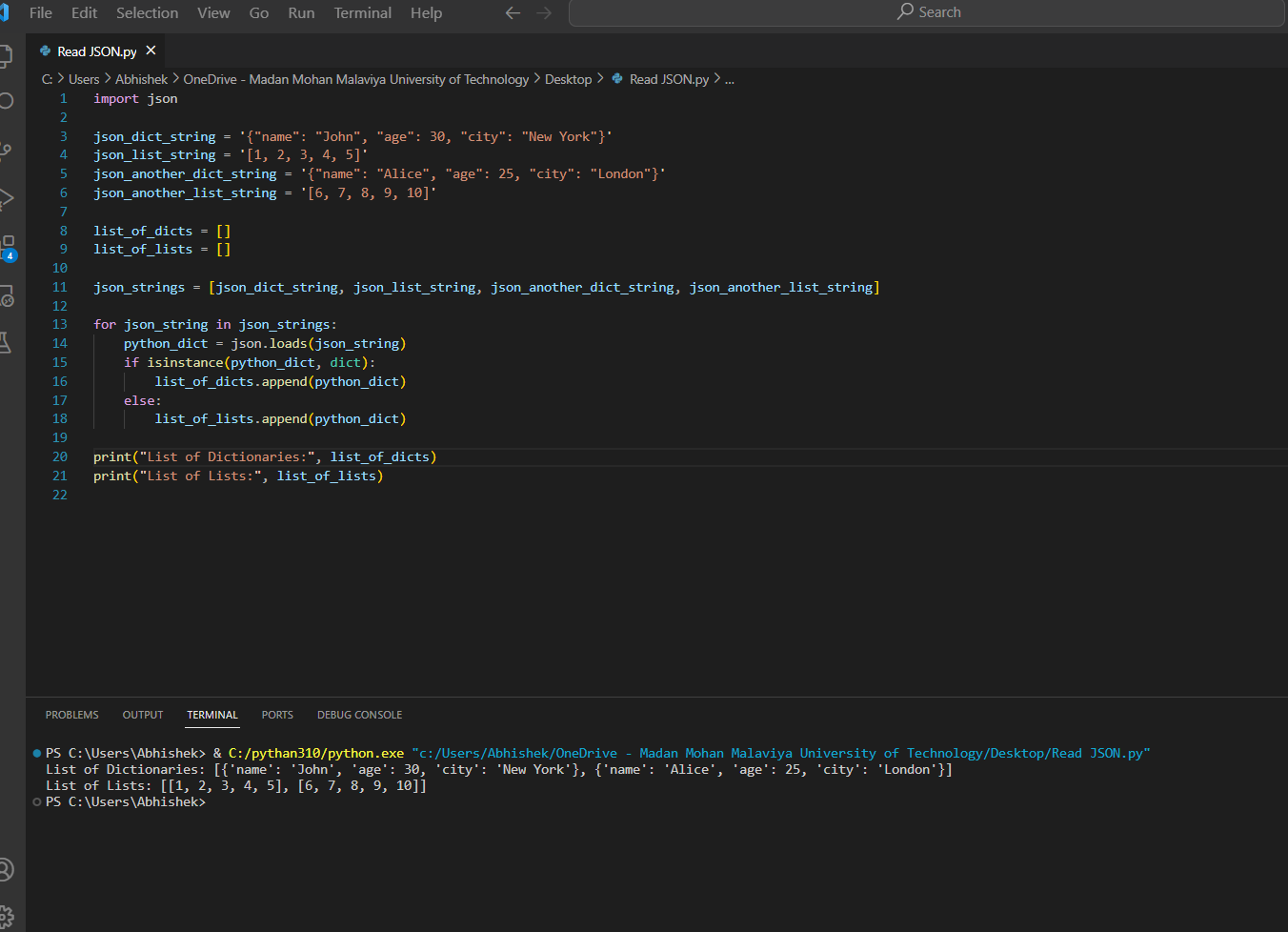
**try:**

**python\_dict = json.loads(json\_dict\_string)**

**except json.JSONDecodeError as e:**

**print("Error decoding JSON:", e)**

This try-except block will catch and print any JSON decoding errors that might occur.  
  
  
**Example:**

****