Assignment No. 9

Submitted By: Sachin Dodake

1. What is a `lambda function` in Python, and how does it differ from a `regular function`?

`Ans-4`

`Lambda Function:`

- A `lambda function` is an anonymous function (i.e., defined without a name) that can take any number of arguments but, unlike normal functions, evaluates and returns only one expression.
- Python `Lambda Functions` are anonymous function means that the function is without a name. As we already know that the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.
- **Syntax:**

```
lambda p1, p2: expression
            p1- parameter1 to pass in lambda function
            p2- parameter2 to pass in lambda function
```

- The anatomy of a lambda function includes three elements:

```
        The keyword lambda — an analog of def in normal functions
        The parameters — support passing positional and keyword arguments, just like
normal functions
        The body — the expression for given parameters being evaluated with the lambda
function
```

In [51]:
```python
# Exaplaination on Lambda function to Multiply argument a with argument b and return the result:

multi = lambda a, b : a * b

print(f"The multiplication using `Lambda Function` is : {multi(5, 6)}")
```

```
The multiplication using `Lambda Function` is : 30
```

Regular Function-Vs-Lambda Function:

- **`Syntax:`** Lambda functions are written in a single line of code, whereas regular functions defined with def can span multiple lines.
- **`Function Name:`** Lambda functions do not have a name, whereas regular functions defined with def have a name.
- **`Return Statement:`** Lambda functions automatically return the result of the expression they evaluate, while regular functions defined with def require an explicit return statement to return a value.
- **`Arguments:`** Both types of functions can take any number of arguments, but lambda functions are typically used for simple, one-line expressions with one or two arguments.
- **`Functionality:`** Regular functions defined with def can include complex logic, including flow control statements (such as if and while), error handling, and more complex calculations. Lambda functions are typically used for simple operations, such as filtering, mapping, or reducing data.
- **Example:** The examples of `Lambda` & `Regular` function are as below:

In [49]:
```python
#  Explaination of Lambda function that adds two numbers
add_lambda = lambda x, y: x + y

# Explaination of Regular function that adds two numbers
def add_def(x, y):
    return x + y

# Calling the functions
print(f"The result using 'Lambda' function is : {add_lambda(2, 3)}")
print(f"\nThe result using 'Regular' function is : {add_def(2, 3)}")
```

```
The result using 'Lambda' function is : 5

The result using 'Regular' function is : 5
```

2. Can a `lambda function` in Python have `multiple arguments`? If yes, how can you define and use them?

`Ans-2`

- `Python lambda function` can be used with multiple arguments and these arguments are used in evaluating an expression to return a single value. A Python lambda function is used to execute an anonymous function, an anonymous meaning function without a name.
- `Python lambda function` can take any number of arguments, but can only have one expression and they can be used wherever function objects are required. Here is the syntax of the lambda.
- To pass multiple arguments in the lambda function, we must mention all the parameters separated by commas.
- **Syntax:**

```
lambda argument, [argument, argument]: expression
```

- We can use multiple arguements in lambda function in the following ways:

```
i) Python Lambda with Two Arguments,
ii) Using Multiple Iterables,
iii) Using String List as Multiple Arguments.
```

`i) Python Lambda with Two Arguments:`

In [2]:
```python
# Lambda example with two arguments
add = lambda x, y : x + y
print(add(10, 20))
```

```
30
```

`ii) Using Multiple Iterables:`

In [3]:
```python
# Create two lists with numbers
numbers1 = [2, 4, 5, 6, 3]
numbers2 = [1, 3, 2, 2, 4]

# Create lambda function that takes two arguments
add_fun = lambda x, y: x+y

# Use lambda with map() function
add_result = list(map(add_fun, numbers1, numbers2))
print(add_result)
```

```
[3, 7, 7, 8, 7]
```

`iii) Using String List as Multiple Arguments`

In [7]:
```python
# Create string list
list_1 = ["A","B","C","D"]
list_2 = ["a","b","c","d"]
print("List-1:",list_1)
print("List-2:",list_2)

# Use lower() function
lower_result = list(map(lambda x,y: x + y, list_1, list_2))
print("\nResult:",lower_result)
```

```
List-1: ['A', 'B', 'C', 'D']
List-2: ['a', 'b', 'c', 'd']

Result: ['Aa', 'Bb', 'Cc', 'Dd']
```

3. How are `lambda functions` typically used in Python? Provide an example use case.

`Ans-3`

- Lambda functions are frequently used with higher-order functions, which take one or more functions as arguments or return one or more functions.
- A lambda function can be a higher-order function by taking a function (normal or lambda) as an argument.
- Python exposes higher-order functions as built-in functions or in the standard library which includes `map(), filter(), functools.reduce(), as well as key functions like sort(), sorted(), min(), and max().`
- The various use cases of `Python Lmbda function` are listed below:

```
i) Sorting Lists with Custom Keys,
ii) Filtering Lists with the filter() Function,
iii) Applying Transformations with the map() Function,
iv) Using Lambda Functions with functools.reduce(),
v) Creating Small, One-Time-Use Functions,
vi) Implementing Simple Event Handlers,
vii) Using Lambda Functions in GUI Programming with Tkinter,
viii) Implementing Custom Comparison Functions for Data Structures,
ix) Using Lambda Functions in Concurrent Programming with ThreadPoolExecutor.
```

`i) Sorting Lists with Custom Keys:`

- Lambda functions can be used as a custom key function when sorting lists, allowing you to sort based on a specific attribute or calculation.
- **Example:**

In [9]:
```python
students = [('Alice', 90), ('Bob', 85), ('Charlie', 92)]
sorted_students = sorted(students, key=lambda x: x[1])
print(sorted_students)
```

```
[('Bob', 85), ('Alice', 90), ('Charlie', 92)]
```

`ii) Filtering Lists with the filter() Function:`

- The filter() function can be used in conjunction with a lambda function to filter a list based on a specific condition.
- **Example:**

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

```
[2, 4, 6, 8]
```

`iii) Applying Transformations with the map() Function:`

- The map() function can be used with a lambda function to apply a transformation to each element in a list.
- **Example:**

In [11]:
```python
numbers = [1, 2, 3, 4, 5]
squares = list(map(lambda x: x**2, numbers))
print(squares)
```

```
[1, 4, 9, 16, 25]
```

`iv) Using Lambda Functions with functools.reduce():`

- The reduce() function from the functools module can be used with a lambda function to apply a binary operation cumulatively to the elements in a list, reducing the list to a single value.
- **Example:**

In [12]:
```python
from functools import reduce

numbers = [1, 2, 3, 4, 5]
product = reduce(lambda x, y: x * y, numbers)
print(product)
```

```
120
```

`v) Creating Small, One-Time-Use Functions:`

- Lambda functions are ideal for creating small, one-time-use functions that don't need a proper function definition.
- **Example:**

In [13]:
```python
max_value = (lambda x, y: x if x > y else y)(5, 7)
print(max_value)
```

```
7
```

4. What are the `advantages and limitations` of lambda functions compared to `regular functions` in Python?

`Ans-4`

`Advantages:`

- Ease of creating the function.
- Variables with few options.

`Disadvantages:`

- Despite Python's popularity as the world's most popular programming language, the Lambda is not user-friendly.
- The Lambda function will be a problem if we need to build a complex function.
- Strictly limited to a single expression.
- Although it cannot access a global variable, it can access the lone local variable.
- The simplicity of the Lambda function, which is a single statement, cannot be acceptable in all circumstances.
- For clarity, most Python functions and modules contain documentation, which is another drawback of the Lambda function.

5. Are `lambda functions` in Python able to access variables defined outside of their own scope? Explain with an example.

`Ans-5`

- Lambda functions have their own local namespace and can not access variables other than those in their parameter list and those in the global namespace.

6. Write a `lambda function` to calculate the `square of a given number`.

`Ans-6`

In [27]:
```python
number = int(input('Please Enter the  Number : '))          # taking inputs from user

output = lambda number: number * number       # applying LAMBDA function

print(f"The square of given number `{number}` is : {output(number)}")      # Printing result
```

```
Please Enter the  Number : 6
The square of given number `6` is : 36
```

7. Create a `lambda function` to find the `maximum value` in a list of integers.

`Ans-7`

In [3]:
```python
input_list = input('Enter elements of a list separated by comma: ') # taking list elements from user

user_list = input_list.split(',')       # preparing list

print(f"\nThe user list is : {user_list}.")       # printing user list

max_value = max(user_list, key=lambda x:int(x))      # defining LAMBDA function

print(f"\nThe maximum number from the user list is : {max_value}.")       # printing max value from list
```

```
Enter elements of a list separated by comma: 11,55,99,33,77,22,44,66,88

The user list is : ['11', '55', '99', '33', '77', '22', '44', '66', '88'].

The maximum number from the user list is : 99.
```

8. Implement a `lambda function` to filter out all the `even numbers` from a list of integers.

`Ans-8`

In [2]:
```python
num_list = [11, 22, 33, 44, 55, 66, 77, 88, 99, 100]     # defining list

print(f"\nThe number list is : {num_list}.")       # printing number list

even_num = list(filter(lambda x: x % 2 == 0, num_list))   # applying LAMBDA function

print(f"\nThe List of Even Numbers is : {even_num}")      # printing list of even numbers
```

```
The number list is : [11, 22, 33, 44, 55, 66, 77, 88, 99, 100].

The List of Even Numbers is : [22, 44, 66, 88, 100]
```

9. Write a `lambda function` to sort a list of strings in `ascending order` based on the length of each string.

`Ans-9`

In [7]:
```python
str_list = ["rohan", "amy", "sapna", "muhammad", "aakash", "raunak", "chinmoy"]       # defining list of str

print(f"\nThe list of string is : {str_list}.")       # printing user list of string

sortedList = sorted(str_list, key=lambda x: len(x))      # defining LAMBDA function

print(f"\nThe list elements in ascending order is : {sortedList}.")       # printing max value from list
```

```
The list of string is : ['rohan', 'amy', 'sapna', 'muhammad', 'aakash', 'raunak', 'chinmoy'].

The list elements in ascending order is : ['amy', 'rohan', 'sapna', 'aakash', 'raunak', 'chinmoy', 'muhamma
d'].
```

10. Create a `lambda function` that takes two lists as input and returns a `new list` containing the `common elements` between the two lists.

`Ans-10`

In [17]:
```python
list_1 = [11,22,33,44,55,66,77,88,99]
list_2 = [22,77,44,99,33]

print(f"The FIRST list is : {list_1}.")
print(f"The SECOND list is : {list_2}.")

list_new = list(filter(lambda x: x in list_1, list_2))

print(f"\nThe NEW List with common elements is : {list_new}")
```

```
The FIRST list is : [11, 22, 33, 44, 55, 66, 77, 88, 99].
The SECOND list is : [22, 77, 44, 99, 33].

The NEW List with common elements is : [22, 77, 44, 99, 33]
```

11. Write a `recursive function` to calculate the factorial of a given `positive integer`.

`Ans-11`

In [21]:
```python
# Factorial of a number using recursion

def recur_factorial(n):
    if n == 1:
        return n
    else:
        return n*recur_factorial(n-1)

num = 7

# check if the number is negative

if num < 0:
    print("Please provide the postive number.")
elif num == 0:
    print("The factorial of 0 is 1.")
else:
    print(f"The factorial of `{num}` is :  {recur_factorial(num)}.")
```

```
The factorial of `7` is :  5040.
```

12. Implement a recursive function to compute the `nth Fibonacci number`.

`Ans-12`

In [28]:
```python
#recursive approach
def fib_num(n):
    if n<=0:
        print("Fibonacci can't be computed")
   # First Fibonacci number
    elif n==1:
        return 0
   # Second Fibonacci number
    elif n==2:
        return 1
    else:
        return fib_num(n-1)+fib_num(n-2)

#input
n=int(input("Enter the Number : "))
print(f"\nThe {n}th Fibonacci number is : {fib_num(n)}")
```

```
Enter the Number : 4

The 4th Fibonacci number is : 2
```

13. Create a recursive function to find the `sum of all the elements` in a given list.

`Ans-13`

In [3]:
```python
my_list=[11,22,33,44,55]

def sum_list(my_list, nSum):
    if len(my_list):
        return sum_list(my_list[1:], nSum+my_list[0])
    else:
        return nSum

print(f"The given list is : {my_list}.")
print(f"\nThe sum of all the elements of given list is : {sum_list(my_list, 0)}.")
```

```
The given list is : [11, 22, 33, 44, 55].

The sum of all the elements of given list is : 165.
```

14. Write a recursive function to determine whether a `given string is a palindrome`.

`Ans-14`

In [6]:
```python
def check_palindrome(my_str):
    if len(my_str) < 1:
        return True
    else:
        if my_str[0] == my_str[-1]:
            return check_palindrome(my_str[1:-1])
        else:
            return False

my_string = str(input("Enter the string :"))

print(f"\nThe given string is : {my_string}.")


if(check_palindrome(my_string)==True):
    print("\nThe string is a PALINDROME.")
else:
    print("\nThe string isn't a PALINDROME.")
```

```
Enter the string :RADAR

The given string is : RADAR

The string is a PALINDROME.
```

15. Implement a recursive function to find the `greatest common divisor (GCD)` of two positive integers

Ans-15

In [9]:
```
def gcd(num1, num2):
    if num1 == num2:
        return num1
    elif num1 < num2:
        return gcd(num2, num1)
    else:
        return gcd(num1, num1 - num2)

num1 = 25
num2 = 45
print(f"The greatest common divisor of {num1} & {num2} is : {gcd(num1, num2)}.")
```

```
The greatest common divisor of 25 & 45 is : 5.
```