



TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A REPORT ON
Use of function in python programming

SUBMITTED BY:
SURENDRA TIRUWA (081BEL091)

SUBMITTED TO:
DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

1. Write a Python function named `greet_user` that takes a user's name and prints: `Hello, <name>! Welcome to Python`. Call the function with a sample name.

CODE:

```
def greet_user(name):  
    print(f'Hello, {name}! Welcome to Python.')  
  
greet_user("Test Name")
```

OUTPUT:

```
Hello, Test Name! Welcome to Python.
```

2. Create a function `power(base, exponent=2)` that returns the result of base raised to the power of exponent. Demonstrate it with and without the exponent argument.

CODE:

```
def power(base, exponent=2):  
    return base ** exponent  
  
print(power(3, 3))  
print(power(4))
```

OUTPUT:

```
27  
16
```

3. Write a function `book_info(title, author, year)` that prints book details. Call the function using keyword arguments in different orders.

CODE:

```
def book_info(title, author, year):  
    print(f'Title: {title}, Author: {author}, Year: {year}')
```

```
book_info(author="Orwell", year=1949, title="1984")
```

OUTPUT:

```
Title: 1984, Author: Orwell, Year: 1949
```

4. Create a function `sum_numbers(*args)` that accepts any number of numeric arguments and returns their sum. Test it with 2, 3, and 5 numbers.

CODE:

```
def sum_numbers(*args):  
    return sum(args)  
  
print(sum_numbers(2, 3))  
print(sum_numbers(2, 3, 5))  
print(sum_numbers(1, 2, 3, 4, 5))
```

OUTPUT:

```
5  
10  
15
```

5. Write a function `student_profile(kwargs)` that prints the key-value pairs passed (e.g., name, age, grade). Call it with at least three named arguments.**

CODE:

```
def student_profile(**kwargs):  
    print(kwargs)  
  
student_profile(name="Sushant", age=20, grade="A")
```

OUTPUT:

```
{'name': 'Sushant', 'age': 20, 'grade': 'A'}
```

6. Write a lambda function to compute the square of a number. Use it to compute the square of 5 and 12.

CODE:

```
square = lambda x: x * x
```

```
print(square(5))
```

```
print(square(12))
```

OUTPUT:

```
25
144
```

7. Given a list of numbers [1, 2, 3, 4, 5], use map() and a lambda function to return a new list with each number doubled.

CODE:

```
nums = [1, 2, 3, 4, 5]
```

```
doubled = list(map(lambda x: x * 2, nums))
```

```
print(doubled)
```

OUTPUT:

```
[2, 4, 6, 8, 10]
```

8. Given a list [10, 15, 20, 25, 30], use filter() and a lambda function to extract numbers divisible by 10.

CODE:

```
nums = [10, 15, 20, 25, 30]
```

```
filtered = list(filter(lambda x: x % 10 == 0, nums))  
print(filtered)
```

OUTPUT:

```
[10, 20, 30]
```

9. Given a list of temperatures in Celsius [36.5, 37.0, 39.2, 35.6, 38.7], convert them to Fahrenheit using map(), Filter out those above 100°F using filter().

CODE:

```
temps_c = [36.5, 37.0, 39.2, 35.6, 38.7]  
temps_f = list(map(lambda c: (c * 9/5) + 32, temps_c))  
above_100 = list(filter(lambda f: f > 100, temps_f))  
print(above_100)
```

OUTPUT:

```
[102.56, 101.66]
```

Mini Project:

Simple To-Do Manager Using Functional Programming

Objective: Manage a list of to-do tasks using functions, lambda, filter, and map.

Requirements:

- Allow adding tasks using a function `add_task(task_list, task_name)`.
- Each task is a dictionary: `{ "name": str, "completed": bool }`.
- Use `lambda` and `filter()` to list only incomplete tasks.
- Use `map()` to mark all tasks as completed.
- Include a `search_tasks(task_list, keyword)` function using `filter()` and `lambda`.

CODE:

```
def add_task(task_list, task_name):  
    task_list.append({"name": task_name, "completed": False})  
    return task_list  
  
def list_pending(tasks):  
    return list(filter(lambda task: not task["completed"], tasks))  
  
def complete_all(tasks):  
    return list(map(lambda task: {**task, "completed": True}, tasks))  
  
def search_tasks(task_list, keyword):  
    return list(filter(lambda task: keyword.lower() in task["name"].lower(), task_list))  
  
tasks = []  
tasks = add_task(tasks, "Buy groceries")  
tasks = add_task(tasks, "Finish assignment")  
tasks = add_task(tasks, "Call friend")  
  
print("Pending Tasks:", list_pending(tasks))  
  
tasks = complete_all(tasks)  
  
print("Search Result:", search_tasks(tasks, "call"))
```

OUTPUT:

```
Pending Tasks: [{'name': 'Buy groceries', 'completed': False}, {'name': 'Finish assignment',  
, 'completed': False}, {'name': 'Call friend', 'completed': False}]  
  
Search Result: [{'name': 'Call friend', 'completed': True}]
```