# Functions - Solutions

## Solution 1: Greeting Function

```python
def greet_user(name):
    print(f"Hello, {name}! Welcome!")
```

---

## Solution 2: Sum of Two Numbers

```python
def sum_two_numbers(a, b):
    return a + b
```

---

## Solution 3: Convert Celsius to Fahrenheit

```python
def celsius_to_fahrenheit(celsius):
    return (9/5) * celsius + 32
```

---

## Solution 4: Calculate Area of Circle

```python
import math

def calculate_circle_area(radius):
    return math.pi * radius ** 2
```

---

## Solution 5: Simple Interest

```python
def calculate_simple_interest(principal, rate, time):
    interest = (principal * rate * time) / 100
    return interest
```

---

## Solution 6: Compound Interest

```python
def calculate_compound_interest(principal, rate, time):
    amount = principal * (1 + rate / 100) ** time
    interest = amount - principal
    return interest, amount
```

---

## Solution 7: Maximum of Three Numbers

```python
def max_of_three(a, b, c):
    return max(a, b, c)
```

---

## Solution 8: Check Even or Odd with Lambda

```
is_even_or_odd = lambda number: 'even' if number % 2 == 0 else 'odd'
print(is_even_or_odd(3))  # Output: odd
print(is_even_or_odd(4))  # Output: even
```

## Solution 9: Sum of Squares using map()

```
numbers = range(1, 11)
squares = list(map(lambda x: x ** 2, numbers))
print(squares)
```

## Solution 10: Filter Positive Numbers

```
def filter_positive(numbers):
    return list(filter(lambda x: x > 0, numbers))

numbers = [-10, 20, -30, 40, -50]
print(filter_positive(numbers))  # Output: [20, 40]
```

## Advanced Solution 1: Recursive Factorial Function

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```

## Advanced Solution 2: Higher-Order Function for Applying Functions

```
def apply_function(func, x):
    return func(func(x))

def double(x):
    return x * 2

print(apply_function(double, 5))  # Output: 20
```

## Advanced Solution 3: Net Present Value (NPV) with Lambda

```
def calculate_npv(cash_flows, rate):
    # Define the lambda function for calculating discounted cash flows
    discount = lambda cf, t: cf / (1 + rate) ** t
    # Use a list comprehension with the lambda function
    return sum([discount(cf, t) for t, cf in enumerate(cash_flows, 1)])

# Example usage:
cash_flows = [-1000, 200, 300, 400, 500]
print("NPV:", calculate_npv(cash_flows, 0.05))  # Output: NPV value
```

## Advanced Solution 4: Zip and Sort

```python
def zip_and_sort(names, scores):
    combined = zip(names, scores)
    return sorted(combined, key=lambda pair: pair[1], reverse=True)

names = ['Alice', 'Bob', 'Charlie']
scores = [85, 90, 95]
print(zip_and_sort(names, scores))
```

---

## Advanced Solution 5: Reduce for Cumulative Product

```python
from functools import reduce

def cumulative_product(numbers):
    return reduce(lambda a, b: a * b, numbers)

numbers = [1, 2, 3, 4]
print(cumulative_product(numbers))  # Output: 24
```