Function Overloading

Function overloading refers to the ability to define multiple functions with the same name but different parameter lists. Function overloading allows you to create functions that behave differently depending on the arguments passed to them. Python Does Not Support Function Overloading: Unlike some other programming languages (e.g., C++), Python does not support function overloading by default. Emulating Function Overloading: In Python, you can emulate function overloading using default parameter values or variable-length argument lists (e.g., `*args`, `**kwargs`).

Example:

```
def greet(name=None):
 if name:
   print(f"Hello, {name}!")
 else:
   print("Hello, world!")

greet() # Output: Hello, world!
greet("Alice") # Output: Hello, Alice!
```

- In this example, the `greet` function behaves differently based on whether the `name` argument is provided or not.

Exercise 1:

Create a function called `calculate` that calculates the area of different shapes based on the number of arguments passed. If one argument is provided, it should calculate the area of a circle (π * r^2), and if two arguments are provided, it should calculate the area of a rectangle (length * width). Test the `calculate` function with different argument combinations.

Answer 1:

```
import math

def calculate(*args):
 if len(args) == 1:
 return math.pi * args[0] ** 2 # Area of circle
 elif len(args) == 2:
 return args[0] * args[1] # Area of rectangle
 else:
 raise ValueError("Unsupported number of arguments")

# Test the function
print("Area of circle:", calculate(5))
print("Area of rectangle:", calculate(3, 4))
```

Exercise 2:
Define a function called `concatenate` that concatenates strings based on the number of arguments passed. If one argument is provided, it should return the same string. If two arguments are provided, it should concatenate them. If three arguments are provided, it should concatenate all three strings. Test the `concatenate` function with different argument combinations.

Answer 2:

```
def concatenate(*args):
 if len(args) == 1:
 return args[0]
 elif len(args) == 2:
 return args[0] + args[1]
 elif len(args) == 3:
 return args[0] + args[1] + args[2]
 else:
 raise ValueError("Unsupported number of arguments")

# Test the function
```

```
print("Concatenated string:", concatenate("Hello"))
print("Concatenated string:", concatenate("Hello", " ", "World"))
```

Exercise 3:
Create a function called `average` that calculates the average of numbers based on the number of arguments passed. If no arguments are provided, it should return 0. If one or more arguments are provided, it should calculate the average. Test the `average` function with different argument combinations.

Answer 3:

```
def average(*args):
 if len(args) == 0:
 return 0
 else:
 return sum(args) / len(args)

# Test the function
print("Average:", average())
print("Average:", average(3, 5, 7))
```