College of Engineering, Software Engineering Department

SE 2143/2133 Software Development III

Laboratory 1

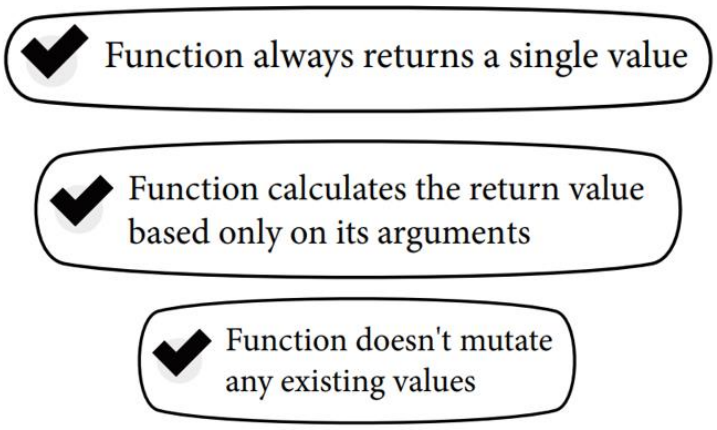
Pure Functions

Objectives:

- Understand the concept of pure functions.
- Learn to identify and refactor impure functions into pure functions.
- Practice applying functional programming principles in Python.

Instructions:

Unless stated otherwise, for each exercise, refactor the given code into pure function(s). Recall that a pure function must fulfill the following three conditions:



Exercise 1 (10 pts)

```
data = [1, 2, 3]

def add_element(element: int) -> None:
    data.append(element)

add_element(4)
print(data)  # Outputs: [1, 2, 3, 4]
```

Exercise 2 (10 pts)

```
user_info = {"name": "Alice", "age": 25}

def update_age(new_age: int) -> None:
    user_info["age"] = new_age

update_age(26)
print(user_info)  # Outputs: {'name': 'Alice', 'age': 26}
```

Exercise 3 (20 pts)

In this exercise, make a pure function named insert_word that takes two arguments – the first arg is a list of English words which are assumed to ALREADY be in alphabetical order (ex. ['cat', 'dog', 'eagle', 'elephant']); the second arg is an English word (ex. 'dragon').

The function should insert the second arg in to the first arg but in the correct position (i.e. alphabetical order is maintained). Therefore, the example args above should produce a new list which is ['cat', 'dog', 'dragon', 'eagle', 'elephant']. Don't forget the correct type annotations!

Exercise 4 (30 pts)

You are managing a system that tracks employee working hours and calculates pay. The `EmployeePayCalculator` class tracks an employee's hours and contains methods for setting hours and calculating the weekly pay. The business rule is: "Overtime (hours worked over 40 in a week) is paid at 1.5 times the regular hourly rate."

Refactor the `EmployeePayCalculator` class so that it no longer maintains state. Instead, refactor it into a pure function that takes in the necessary data (hourly rate, hours worked) and returns the calculated pay based on the business rule.

```
class EmployeePayCalculator:
    def __init__(self, hourly_rate: float):
        self._hourly_rate: float = hourly_rate
        self._hours_worked: float = 0

    def set_hours_worked(self, hours: float) -> None:
        self._hours_worked = hours

    def calculate_pay(self) -> float:
        if self._hours_worked > 40:
            overtime_hours = self._hours_worked - 40
            regular_pay = 40 * self._hourly_rate
            overtime_pay = overtime_hours * self._hourly_rate * 1.5
            return regular_pay + overtime_pay
        else:
            return self._hours_worked * self._hourly_rate

employee = EmployeePayCalculator(20)
employee.set_hours_worked(45)
print(employee.calculate_pay())   # Outputs: 950.0

employee.set_hours_worked(16)
print(employee.calculate_pay())   # Outputs: 320
```

Exercise 5 (30 pts)

You are developing an e-commerce platform that handles shopping carts. The ShoppingCart class tracks the items added to the cart and calculates the total cost. The business rule is: "If the total cost exceeds $100, a 10% discount is applied."

Refactor the ShoppingCart class so that it no longer maintains state. Instead, refactor it into pure functions that take in the necessary data (items) and return the calculated total cost based on the business rule.

```
class ShoppingCart:
    def __init__(self):
        self._items: list = []
        self._total_cost: float = 0
        self._discount_percent: float = 0.0

    def add_item(self, name: str, price: float) -> None:
        self._items.append({"name": name, "price": price})
        self._total_cost += price

        if self._total_cost > 100:
            self._discount_percent = 0.1  # 10% discount
        else:
            self._discount_percent = 0.0

    def calculate_total(self) -> float:
        return self._total_cost * (1 - self._discount_percent)

cart = ShoppingCart()
cart.add_item("Laptop", 90)
cart.add_item("Mouse", 20)
print(cart.calculate_total())  # Outputs: 99.0 (with discount applied)
```

Web submission

Submit a .zip file containing five files named 1.py, 2.py, 3.py, 4.py, 5.py, containing the solutions to the exercises above.

There are no naming conventions for the .zip file. Just use both of your surnames to fill up the "First name" and "Last name" fields when uploading.

For example, if I am partners with Mr. Ardiente. We would put "Ng" and "Ardiente" on the fields when uploading on dropbox.

Submit to https://www.dropbox.com/request/k5rUoSJsDbgSlyAw7PvZ