# Data Structures

Homework Assignment 1 - Python and OOP

### Tasks on this Worksheet

- Problem 1 - Introduce Yourself (OOP) - 25 Points
- Problem 2 - Vehicle Class - 25 Points
- Problem 3 - Custom Generator - 25 Points
- Problem 4 - Flip the Bit - 25 Points

### Notes and Requirements

- Your submission must be your effort. You can not copy other students' code.
- This worksheet is graded on performance; Implementations must be correct.
- You are encouraged to visit our office hours to ask coding questions.
- Only the latest (most recent) submission is graded.
- Late submissions are not considered for grading.

**All assignments on this worksheet are manually graded.**

Fall 2024 - Revision 1.1 - *January*

# Problem 1 - Introduce Yourself (OOP) - 25 Points

This problem will familiarize you with Python classes and basic object-oriented programming concepts. Create a class called ***Person*** that represents a person with the following attributes:

- first_name (string)
- last_name (string)
- hobby (string)

Implement a method *introduce()* that returns a string introducing the person, e.g., *"Hi, my name is John Doe. I like playing guitar."*

### Requirements

- You need to implement all required attributes and methods.
- Object-oriented design principles need to be applied correctly.
- You need to assign meaningful parameter and variable names.
- Instantiate at least three persons.

### Hint

- You can implement additional methods and attributes to familiarise yourself with OOP.

### Example Implementation

```
p1 = Person("John", "Doe", 20, "playing guitar")
print(p1.introduce())
# Output: Hi, my name is John Doe. I like playing guitar.
```

# Problem 2 - Vehicle Class - 25 Points

This problem aims to explore inheritance in OOP by modeling a hierarchy of vehicles.

Create a base class called ***Vehicle*** with attributes:

- make (string)
- model (string)
- year (integer)

Implement a method *get_description()* that returns a formatted string like *"2021 Toyota Corolla."*

Create a subclass ***Car*** that inherits from ***Vehicle*** and adds:

- *doors (integer)*

Create another subclass ***Truck*** that inherits from ***Vehicle*** and adds:

- *payload_capacity (float)*

### Important

- Object-oriented design principles need to be applied correctly.
- You need to assign meaningful parameter and variable names.

### Hint

- Try to draw a diagram to sketch the relationship between the classes.

### Example Implementation

```
car = Car("Toyota", "Corolla", 2021, 4)
print(car.get_description())
# Output: 2021 Toyota Corolla, 4-door

truck = Truck("Ford", "F-150", 2020, 1.5)
print(truck.get_description())
# Output: 2020 Ford F-150, Payload capacity: 1.5 tons
```

# Problem 3 - Custom Generator - 25 Points

Generators in Python provide a simple way to create iterators. In this problem, you'll create a custom generator. Write a function *custom_generator(n)* that yields the numbers from 0 to n-1, but for numbers divisible by 3, yield the string *"Fizz"* instead of the number, and for numbers divisible by 5, yield the string *"Buzz"*.

### Important

- You can not use any libraries.
- The input can be any positive integer.

### Example Implementation

```
gen = custom_generator(10)
print(list(gen))
# Output: [0, 1, 2, 'Fizz', 4, 'Buzz', 'Fizz', 7, 8, 'Fizz']
```

# Problem 4 – Flip the Bit - 25 Points

This problem will test your understanding of bitwise operations in Python. Write a function *flip_bit(num, bit_position)* that flips (*inverts*) the bit at the specified position in a given number. Assume bit_position starts from 0 (the rightmost bit is position 0).

### Requirements

- You can only use bitwise operations.
- You are not allowed to cast (*change the type of*) parameters.
- You are not allowed to use Python functions such as *`bin()`*.

### Example Implementation

```
print(flip_bit(8, 1))  # Output: 10 (Binary: 1000 -> 1010)
print(flip_bit(10, 1))  # Output: 8 (Binary: 1010 -> 1000)
```