# Amit Divekar | Assignment 2: Set C

## Functions, Iterators and Generators - Set C

### Q1. Create a function employee_details(name, **info) that accepts an employee name and any number of keyword arguments (e.g., department, salary) and prints them.

In [1]:
```
def employee_details(name, **info):
    print(f"Employee Name: {name}")
    for key, value in info.items():
        print(f"{key}: {value}")
    print()

employee_details("Amit", department="IT", salary=50000, experience=5)
employee_details("Priya", department="HR", salary=45000, location="Mumbai")
```

```
Employee Name: Amit
department: IT
salary: 50000
experience: 5

Employee Name: Priya
department: HR
salary: 45000
location: Mumbai
```

### Q2. Write a function analyze_string(s) that returns the number of vowels, consonants, and digits in the string.

In [2]:
```
def analyze_string(s):
    vowels = 0
    consonants = 0
    digits = 0

    for char in s:
        if char.isalpha():
            if char.lower() in 'aeiou':
                vowels += 1
            else:
                consonants += 1
        elif char.isdigit():
            digits += 1
    return vowels, consonants, digits

string = "Hello World 123"
vowels, consonants, digits = analyze_string(string)
print(f"String: {string}")
print(f"Vowels: {vowels}")
print(f"Consonants: {consonants}")
print(f"Digits: {digits}")
```

```
String: Hello World 123
Vowels: 3
Consonants: 7
Digits: 3
```

# Q3. Combine *args and **kwargs in a single function and call it with mixed arguments.

In [3]:
```
def mixed_arguments(*args, **kwargs):
    print("Positional arguments (*args):")
    for arg in args:
        print(arg)

    print("\nKeyword arguments (**kwargs):")
    for key, value in kwargs.items():
        print(f"{key}: {value}")

mixed_arguments(1, 2, 3, "Hello", name="Amit", age=21, city="Mumbai")
```

```
Positional arguments (*args):
1
2
3
Hello

Keyword arguments (**kwargs):
name: Amit
age: 21
city: Mumbai
```

# Q4. Implement a custom iterator class for generating permutations of a string.

In [4]:
```
from itertools import permutations

class StringPermutations:
    def __init__(self, string):
        self.string = string
        self.perms = permutations(string)

    def __iter__(self):
        return self
    def __next__(self):
        perm = next(self.perms)
        return ''.join(perm)

string = "ABC"
print(f"Permutations of '{string}':")
perms = StringPermutations(string)
for perm in perms:
    print(perm)
```

```
Permutations of 'ABC':
ABC
ACB
BAC
BCA
CAB
CBA
```

## Q5. Create a generator pipeline to process and filter a stream of sensor data.

In [5]:
```
def sensor_data_generator():
    data = [25.5, 30.2, 22.8, 35.6, 28.3, 31.9, 26.7, 33.4]
    for value in data:
        yield value

def filter_temperature(data):
    for value in data:
        if value > 25 and value < 35:
            yield value

def convert_to_fahrenheit(data):
    for value in data:
        yield (value * 9/5) + 32

print("Filtered and converted sensor data:")
pipeline = convert_to_fahrenheit(filter_temperature(sensor_data_generator()))
for temp in pipeline:
    print(f"{temp:.2f}°F")
```

```
Filtered and converted sensor data:
77.90°F
86.36°F
82.94°F
89.42°F
80.06°F
92.12°F
```

## Q6. Analyze the memory efficiency of generators versus lists using sys.getsizeof().

In [6]:
```
import sys
```

```
def generator_numbers(n):
    for i in range(n):
        yield i

def list_numbers(n):
    return [i for i in range(n)]

n = 10000

gen = generator_numbers(n)
lst = list_numbers(n)

print(f"Memory size of generator: {sys.getsizeof(gen)} bytes")
print(f"Memory size of list: {sys.getsizeof(lst)} bytes")
print(f"\nMemory saved by using generator: {sys.getsizeof(lst) - sys.getsize
```

```
Memory size of generator: 200 bytes
Memory size of list: 85176 bytes

Memory saved by using generator: 84976 bytes
```