Custom Learnings

Day 14

Python

Function Composition:

```
In [5]: def volume_of_gas(pressure, temperature, gas_constant,):
            """
            09/19/2023 - Pooja Verma - Created function to calculate volume of the gas.
            Calculate the volume of a gas using the ideal gas law.
            Args:
                pressure(float): Pressure in Pascals(Pa).
                temperature(float) : Temperature in Kelvin(K).
                gas_constant(float) : Gas constant for the specific gas.
            Returns:
                float: Volume in cubic meters(m^3).
            """
            return (pressure * 1.0)/(gas_constant * temperature)
        gas_vol = volume_of_gas(10000, 300, 8.314)
        print(f"Gas Volume : {gas_vol} m^3")

        Gas Volume : 4.009301579664823 m^3
```

```
In [9]: def calculate_gas_mass(pressure, temperature, gas_constant, molar_mass):
            """
            09/19/2023 - Pooja Verma - Created function to calculate mass of the gas.
            Calculate the mass of a gas using the ideal gas law and molar mass.
            Args:
                pressure(float): Pressure in Pascals(Pa).
                temperature(float) : Temperature in Kelvin(K).
                gas_constant(float) : Gas constant for the specific gas.
                molar_mass(float) : molar mass of the gas in g/mol.
            Returns:
                float: Mass in grams(g).
            """
            volume = volume_of_gas(pressure, temperature, gas_constant)
            return (volume * molar_mass)*1000
        gas_mass = calculate_gas_mass(10000, 321, 8.314, 2.0)
        print(f"Mass of Gas : {gas_mass} g")

        Mass of Gas : 7494.021644233313 g
```

Recursive function:

In [10]:
```
def factorial(n):
    """
    09/19/2023 - Pooja Verma - Created a recursive function to calculate factorial.
    Args:
        n (int): number to calculate factorial.
    Returns:
        int: factorial of the number.
    """
    if n==0: #base condition
        return 1
    else:
        return n*factorial(n-1) #recursive call
factorial(10)
```

Out[10]:
```
3628800
```

In [11]:
```
def calculate_total_depth(segments):
    """
    09/19/2023 - Pooja Verma - Created a recursive function to calculate the sum of segment depths.
    Args:
        segments (list): List of segment depths for segment depth calculation.
    Returns:
        int: sum of segment depths.
    """
    if not segments:
        return 0
    else:
        curr_seg_depth = segments[0]
        remaining_seg = segments[1:]
        return curr_seg_depth + calculate_total_depth(remaining_seg)
calculate_total_depth([1,2,3,4,5,6])
```

Out[11]:
```
21
```

Generator Function:

In [12]:
```
def generate_squares(n):
    """
    09/19/2023 - Pooja Verma - Created a generator function to calculate the square of numbers.
    Args:
        n (int): Number upto which we calculate squares.
    Returns:
        int: saves each number's square.
    """
    for i in range(1, n+1):
        yield i ** 2
generate_squares(5)
```

Out[12]:
```
<generator object generate_squares at 0x7f701055a880>
```

In [13]:
```
for i in generate_squares(5):
    print(i)
```

```
1
4
9
16
25
```

In [15]:
```
#calculate monthly oil production from yearly oil production
def oil_production_m(yearly_value):
    """
    09/19/2023 - Pooja Verma - Created a generator function to calculate the
    monthly value of oil production from yearly value of oil production .
    Args:
    yearly_value (int): Number for whole year oil production.
    Returns:
    int : saves montly oil production in key-value form.
    """
    monthly_value = yearly_value//12
    months = ["jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec"]
    for month in months:
        yield month, monthly_value
for month, production in oil_production_m(12000):
    print(f"{month}:{production}")
```

```
jan:1000
feb:1000
mar:1000
apr:1000
may:1000
jun:1000
jul:1000
aug:1000
sep:1000
oct:1000
nov:1000
dec:1000
```

Decorator Function:

In [21]:
```python
def my_dec(fun):
    def wrapper():
        print("i am starting")
        fun()
        print("i am completed")
    return wrapper
@my_dec
def ida_hello():
    print("Hello")
```

In [22]:
```python
ida_hello()
```

```
i am starting
Hello
i am completed
```

In [26]:
```python
import logging

def my_decorator(fun):
    def wrapper(*args,**kwargs):
        logging.info(f"Calling the function: {fun.__name__}")
        result = fun(*args,**kwargs)
        logging.info(f"{fun.__name__} completed")
        return result
    return wrapper

@my_decorator
def calculate_total_depth(segments):
    if not segments:
        return 0
    else:
        curr_seg_depth = segments[0]
        remaining_seg = segments[1:]
        return curr_seg_depth + calculate_total_depth(remaining_seg)
```

In [27]:
```python
calculate_total_depth([100,200,300])
```

Out[27]:
```
600
```

Positional Arguments: When we simply pass the arguments while calling the function without specifying the parameter name.

Keyword Arguments: When we explicitly mention the parameter names while passing the arguments in function calling.

In [28]:
```python
def calculate_energy_content(composition):
    lhv = 0
    for gas, percentage in composition.items():
        #LHV values for common gases(in J/Kg)
        lhv_values = {
            "methane": 50000,
            "ethane": 48000,
            "propane": 46000,
            "butane": 45000,
        }
        if gas in lhv_values:
            lhv += lhv_values[gas] * (percentage/100)
    return lhv
```

In [29]:
```python
gas_composition = {"methane": 80, "ethane": 10, "propane": 5, "butane": 5}
```

Class:

In [4]:
```
class Petro_Corp:
    def __init__(self):
        print("")

    def calculate_total_depth(self,segments):
        if not segments:
            return 0
        else:
            curr_seg_depth = segments[0]
            remaining_seg = segments[1:]
            return curr_seg_depth + self.calculate_total_depth(remaining_seg)

    def volume_of_gas(self,pressure, temperature, gas_constant,):
        return (pressure * 1.0)/(gas_constant * temperature)

    def calculate_gas_mass(self,pressure, temperature, gas_constant, molar_mass):
        volume = volume_of_gas(pressure, temperature, gas_constant)
        return (volume * molar_mass)*1000

    def oil_production_m(self,yearly_value):
        monthly_value = yearly_value//12
        months = ["jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov", "dec"]
        for month in months:
            yield month, monthly_value
```

In [5]:
```
pc = Petro_Corp()
pc.calculate_total_depth([1,2,3])
```

Out[5]: 6

Timestamps:

In [10]:
```
import time
curr_timestamp = time.time()
print(curr_timestamp)

test = datetime.fromtimestamp(curr_timestamp).strftime('%Y-%m-%d')
print(test)
```

```
1695113710.1944788
2023-09-19
```

In [9]:
```
from datetime import datetime
curr_datetime = datetime.now()
print(curr_datetime)
```

```
2023-09-19 08:55:05.603198
```

In [16]:
```
from datetime import datetime
curr_datetime = datetime.now().strftime('%H')
print(curr_datetime)
```

```
08
```

In [17]:
```
curr_datetime = datetime.now().strftime('%h')
print(curr_datetime)
```

```
Sep
```

In [18]:
```
curr_datetime = datetime.now().strftime("%Y")
print(curr_datetime)
```

```
2023
```

In [19]:
```
curr_datetime = datetime.now().strftime("%y")
print(curr_datetime)
```

```
23
```

In [20]:
```
curr_datetime = datetime.now().strftime("%M")
print(curr_datetime)
```

```
01
```

In [21]:
```
curr_datetime = datetime.now().strftime("%m")
print(curr_datetime)
```

```
09
```

Exception Handling:

```
In [23]: a=10
         b=0
         try:
             result = a/b
             print(result)
         except:
             print("Error : Someone divided by zero")

         Error : Someone divided by zero
```

List Comprehension:

```
In [25]: a=[]
         for i in range(1,6):
             a.append(i**2)
         print(a)

         [1, 4, 9, 16, 25]
```

```
In [27]: a=[i**2 for i in range(1,6)]
         print(a)

         [1, 4, 9, 16, 25]
```

```
In [28]: even_number=[]
         for i in range(1,10):
             if i%2 == 0:
                 even_number.append(i)
         print(even_number)

         [2, 4, 6, 8]
```

```
In [29]: even_number = [i for i in range(1,10) if i%2==0]
         print(even_number)

         [2, 4, 6, 8]
```

Lambda Function:

```
In [31]: add = lambda a,b : a+b
         print(add(5,6))

         11
```