Assignments 2

```
t='Hello'
print(t[0].lower()+t[1:5])
s='here'
print(s)
w='LOVELY'
print(w.lower())
p='HeLLo WoRLd'
print(p[0].lower()+p[1].upper()+p[2:4].lower()+p[4].upper()+' '+p[6].lower()+p[7].upper()+p[8:10].lower()+p[10].upper())
p='HelloWorld'
print(p[1:5]+p[6:9])
def count_upper_lower(s):
    upper=0
    lower=0
    for char in s:
        if char.isupper():
            upper+=1
        elif char.islower():
            lower+=1
    return upper, lower
s='EngiNEEr'
upper, lower=count_upper_lower(s)
print(f'Uppercase:{upper}, Lowercase:{lower}')
import re
def remove_non_letters(input_string):
    return re.sub(r'[^a-zA-Z]','',input_string)
input_str='Data-Driven@2025!'
output_str=remove_non_letters(input_str)
```

```
print(output_str)
a=3
b=4
c=5
print(float(b-a+c))
```

Assignment 4

```
import random
def guessing_game():
    num_to_guess=random.randint(1,20)
    attempt=0
    print('welcome to a guessing game!')
    print('guess a num from between 1 to 20')
    while True:
        try:
            guess=int(input('enter a num: '))
            attempt+=1
            if  guess < num_to_guess:
                print('the num guess is too low')
            elif guess > num_to_guess:
                print('the num guess is too high')
            else:
                print(f'congratulation you have guess from the the {attempt} attempt')
                break
        except ValueError:
```

```
        print('please enter a valid num.')
guessing_game()
```

assignment 1

task 1

```
String = "shaaban"
Intger = 50
Float = 19.896
Boolean = False

print(f" {String} : ", "Type ",{type(String)})
print(f" {Intger} : ", "Type" , {type(Intger)})
print(f" { Float} : ", "Type", {type( Float)})
print(f" { Boolean} : ", "Type", {type( Boolean)})
```

task 2

```
floatNum = 19.99
intNum = 50
SNum = '50'


float_to_int = int(floatNum)
int_to_string = str(intNum)
str_to_float = float(SNum)

print("Float to Integer: " , float_to_int)
print("Integer to String: " , int_to_string)
print("String to float: " , str_to_float)
```

task 3

```
print("Enter your firstName: ")
fname =input()

lName = input("Enter your lastName: ")

print(f"Hello {fname} {lName}")
```

task 4

```
age = 20

print("You are" , age , " years old")
```

task 5

```
i=0
word  = input("Enter you favourite word: ")

nTimes= int(input("Enter the number of times to reapeat:"))

while i <nTimes:
    print(word)
    i=i+1
pass
```

assignment 3

```
def multiplication_table(n):
    print(f'multiplication table for {n}')
    for i in range(1,12):
        print(f'{n}*{i}={n*i}')
num=int(input('enter a num:'))
multiplication_table(num)
```

Assignment 6

```
# petroleum_formulas.py

# This program uses Object-Oriented Programming (OOP)
# to model 6 important formulas used in Petroleum Engineering.

# Each formula is represented as a class with a calculate() method.
# All formula classes inherit from a base class for structure and reusability.

# ---------------------
# Base Formula Class
# ---------------------
class PetroleumFormula:
    def calculate(self):
        # Every child class must implement its own calculate method
        raise NotImplementedError("This formula needs a calculate() method.")

# ---------------------
# 1. Ideal Gas Law: P = nRT / V
# ---------------------
class IdealGasLaw(PetroleumFormula):
    def _init_(self, moles, gas_constant, temperature, volume):
        self.n = moles
        self.R = gas_constant
        self.T = temperature
        self.V = volume

    def calculate(self):
        try:
            pressure = (self.n * self.R * self.T) / self.V
            return round(pressure, 2)
        except ZeroDivisionError:
            return "Error: Volume cannot be zero."


# ---------------------
# 2. Darcy’s Law: Q = (kAΔP) / (μL)
# ---------------------
class DarcysLaw(PetroleumFormula):
    def _init_(self, permeability, area, pressure_drop, viscosity, length):
        self.k = permeability
        self.A = area
        self.deltaP = pressure_drop
        self.mu = viscosity
        self.L = length

    def calculate(self):
        try:
            flow_rate = (self.k * self.A * self.deltaP) / (self.mu * self.L)
            return round(flow_rate, 4)
        except ZeroDivisionError:
            return "Error: Viscosity and length cannot be zero."
```

```
# ---------------------
# 3. Hydrostatic Pressure: P = ρgh
# ---------------------
class HydrostaticPressure(PetroleumFormula):
    def _init_(self, fluid_density, gravity, height):
        self.rho = fluid_density
        self.g = gravity
        self.h = height

    def calculate(self):
        pressure = self.rho * self.g * self.h
        return round(pressure, 2)

# ---------------------
# 4. Oil Formation Volume Factor: Bo = V_res / V_std
# ---------------------
class FormationVolumeFactor(PetroleumFormula):
    def _init_(self, reservoir_volume, standard_volume):
        self.V_res = reservoir_volume
        self.V_std = standard_volume

    def calculate(self):
        try:
            Bo = self.V_res / self.V_std
            return round(Bo, 3)
        except ZeroDivisionError:
            return "Error: Standard volume cannot be zero."

# ---------------------
# 5. API Gravity: API = (141.5 / SG) - 131.5
# ---------------------
class APIGravity(PetroleumFormula):
    def _init_(self, specific_gravity):
        self.sg = specific_gravity

    def calculate(self):
        try:
            api = (141.5 / self.sg) - 131.5
            return round(api, 2)
        except ZeroDivisionError:
            return "Error: Specific gravity cannot be zero."

# ---------------------
# 6. Productivity Index: PI = Q / (P_res - P_wf)
# ---------------------
class ProductivityIndex(PetroleumFormula):
    def _init_(self, flow_rate, reservoir_pressure, wellbore_pressure):
        self.Q = flow_rate
        self.P_res = reservoir_pressure
        self.P_wf = wellbore_pressure

    def calculate(self):
        try:
            PI = self.Q / (self.P_res - self.P_wf)
            return round(PI, 3)
        except ZeroDivisionError:
            return "Error: Reservoir pressure and wellbore pressure cannot be equal."
```

```
# ---------------------
# Polymorphic function to display results
# ---------------------
def print_formula_result(formula_object: PetroleumFormula, formula_name: str):
    print(f"🔹 {formula_name} Result: {formula_object.calculate()}")


# ---------------------
# Main Program
# ---------------------
if _name_ == "_main_":
    print("🔬 Welcome to the Petroleum Engineering Formula Calculator!\n")

    # Create objects for each formula with sample values
    formulas = [
        ("Ideal Gas Law", IdealGasLaw(moles=1, gas_constant=8.314, temperature=350, volume=22.4)),
        ("Darcy’s Law", DarcysLaw(permeability=100, area=50, pressure_drop=10, viscosity=1.2, length=30)),
        ("Hydrostatic Pressure", HydrostaticPressure(fluid_density=1000, gravity=9.81, height=300)),
        ("Formation Volume Factor", FormationVolumeFactor(reservoir_volume=1.25, standard_volume=1)),
        ("API Gravity", APIGravity(specific_gravity=0.85)),
        ("Productivity Index", ProductivityIndex(flow_rate=500, reservoir_pressure=3000, wellbore_pressure=1000))
    ]

    # Loop through each formula and print results
    for name, formula in formulas:
        print_formula_result(formula, name)

    print("\n✅ All calculations completed.")
```

exercise 1

```
print("hello")
name=input("what is your name? ")
("what problem do you have?")
print("enter a num to test if even or odd")
num=intsa(input("enter num"))
if num%2==0:
 print("number is even")
 print("you are gooded")
 print("thank you have a wonderful day")
else:
 print("number is odd")
 print("you are a cow go and sleep")
 print("have a nice day")
```