```
print("ASSIGNMENT 1")
```

```
ASSIGNMENT 1
```

Task 1 a.

```
int(20)
```

```
20
```

```
str("50")
```

```
'50'
```

```
float(10.5)
```

```
10.5
```

```
bool(True)
```

```
True
```

b.

```
type(20)
```

```
int
```

```
type('50')
```

```
str
```

```
type(10.5)
```

```
float
```

```
type(True)
```

```
bool
```

Task 2 a.

```
int(19.99)
```

```
19
```

b

```
str(50)
```

```
'50'
```

c.

```
float("50")
```

```
50.0
```

d.

```
print(19)
```

```
19
```

```
type(19)
```

```
int
```

```
print('50')
```

```
50
```

```
type("50")
```

```
str
```

```
print(50.0)
```

```
50.0
```

```
type(50.5)
```

```
float
```

Task 3 a

```
firstname = input("Enter your first name ")
lastname = input("Enter your last name ")
```

```
Enter your first name Zakaria
Enter your last name Abdul-Baqi
```

b

```
print(f"Hello {firstname} {lastname}!")
```

```
Hello Zakaria  Abdul-Baqi !
```

Task 4 a

```
age = 20
print("You are "+ str(age) + " years old")
```

```
You are 20 years old
```

b

```
print("Because you can't concatenate an integer and strings")
```

```
Because you can't concatenate an integer and strings
```

Task 5 a

```
input("What's your favourite word ")
```

```
What's your favourite word Sankara
'Sankara '
```

b

```
input("How many times would you like to repeat it ")
```

```
How many times would you like to repeat it I would like to repeat it 100 times
'I would like to repeat it 100 times'
```

c

```
print(("Sankara" + "\t")*10)
```

```
Sankara Sankara Sankara Sankara Sankara Sankara Sankara Sankara Sankara Sankara
```

Task 6 a

```
print("a. => C. TSyntaxErro")
```

```
a. => C. TSyntaxErro
```

```
print("b.=> A. ValueErro")
```

```
b.=> A. ValueErro
```

```
print("c.=> B. TypeError")
```

```
c.=> B. TypeError
```

ASSIGNMENT _4

```
class PetroleumFormula:
    def calculate(self):
        raise NotImplementedError("Subclasses must implement calculate()")

class DarcysLaw(PetroleumFormula):
    def __init__(self, permeability, area, pressure_diff, viscosity, length):
        self.permeability = permeability
        self.area = area
        self.pressure_diff = pressure_diff
        self.viscosity = viscosity
        self.length = length

    def calculate(self):
        try:
            if any(val <= 0 for val in [self.permeability, self.area, self.pressure_diff, s
                raise ValueError("All parameters must be positive")
            flow_rate = (self.permeability * self.area * self.pressure_diff) / (self.viscos
            return flow_rate
        except ValueError as e:
            return f"Error in Darcy's Law: {e}"
        except TypeError:
            return "Error: All parameters must be numeric values"

class MaterialBalanceOil(PetroleumFormula):
    def __init__(self, initial_pressure, current_pressure, initial_compressibility, oil_vol
        self.initial_pressure = initial_pressure
        self.current_pressure = current_pressure
        self.initial_compressibility = initial_compressibility
        self.oil_volume = oil_volume

    def calculate(self):
        try:
```

```
            if self.initial_pressure <= self.current_pressure:
                raise ValueError("Initial pressure must be greater than current pressure")
            if any(val <= 0 for val in [self.initial_compressibility, self.oil_volume]):
                raise ValueError("Compressibility and oil volume must be positive")
            pressure_diff = self.initial_pressure - self.current_pressure
            expansion = self.initial_compressibility * pressure_diff * self.oil_volume
            return expansion
        except ValueError as e:
            return f"Error in Material Balance: {e}"
        except TypeError:
            return "Error: All parameters must be numeric values"

class VogelsIPR(PetroleumFormula):
    def __init__(self, reservoir_pressure, current_pressure, max_flow_rate):
        self.reservoir_pressure = reservoir_pressure
        self.current_pressure = current_pressure
        self.max_flow_rate = max_flow_rate

    def calculate(self):
        try:
            if any(val <= 0 for val in [self.reservoir_pressure, self.max_flow_rate]):
                raise ValueError("Reservoir pressure and max flow rate must be positive")
            if self.current_pressure < 0:
                raise ValueError("Current pressure cannot be negative")
            if self.current_pressure > self.reservoir_pressure:
                raise ValueError("Current pressure cannot exceed reservoir pressure")
            ratio = self.current_pressure / self.reservoir_pressure
            flow_rate = self.max_flow_rate * (1 - 0.2 * ratio - 0.8 * ratio**2)
            return flow_rate
        except ValueError as e:
            return f"Error in Vogel's IPR: {e}"
        except TypeError:
            return "Error: All parameters must be numeric values"

class ArpsDecline(PetroleumFormula):
    def __init__(self, initial_rate, decline_rate, time):
        self.initial_rate = initial_rate
        self.decline_rate = decline_rate
        self.time = time

    def calculate(self):
        try:
            if any(val < 0 for val in [self.initial_rate, self.decline_rate, self.time]):
                raise ValueError("Rates and time cannot be negative")
            if self.decline_rate == 0:
                raise ValueError("Decline rate cannot be zero")
            flow_rate = self.initial_rate * (1 + self.decline_rate * self.time)**(-1)
            return flow_rate
        except ValueError as e:
            return f"Error in Arps Decline: {e}"
        except TypeError:
            return "Error: All parameters must be numeric values"

class STOIIPVolumetric(PetroleumFormula):
    def __init__(self, area, thickness, porosity, saturation, fvf):
        self.area = area
        self.thickness = thickness
        self.porosity = porosity
        self.saturation = saturation
        self.fvf = fvf
```

```
    def calculate(self):
        try:
            if any(val <= 0 for val in [self.area, self.thickness, self.porosity, self.satu
                raise ValueError("All parameters must be positive")
            if self.saturation >= 1:
                raise ValueError("Saturation must be less than 1")
            stoiip = (self.area * self.thickness * self.porosity * (1 - self.saturation)) /
            return stoiip
        except ValueError as e:
            return f"Error in STOIIP Calculation: {e}"
        except TypeError:
            return "Error: All parameters must be numeric values"

class GorRelationship(PetroleumFormula):
    def __init__(self, solution_gor, pressure, bubble_point):
        self.solution_gor = solution_gor
        self.pressure = pressure
        self.bubble_point = bubble_point

    def calculate(self):
        try:
            if any(val <= 0 for val in [self.solution_gor, self.bubble_point]):
                raise ValueError("Solution GOR and bubble point must be positive")
            if self.pressure < 0:
                raise ValueError("Pressure cannot be negative")
            if self.pressure > self.bubble_point:
                gor = self.solution_gor
            else:
                gor = self.solution_gor * (self.pressure / self.bubble_point)
            return gor
        except ValueError as e:
            return f"Error in GOR Calculation: {e}"
        except TypeError:
            return "Error: All parameters must be numeric values"

# Example usage with polymorphism
formulas = [
    DarcysLaw(0.1, 10000, 500, 2, 200),
    MaterialBalanceOil(5000, 3000, 0.0005, 1000000),
    VogelsIPR(3000, 2000, 1000),
    ArpsDecline(1000, 0.1, 5),
    STOIIPVolumetric(1000000, 50, 0.2, 0.3, 1.5),
    GorRelationship(500, 1500, 2000)
]

for formula in formulas:
    result = formula.calculate()
    print(f"{formula.__class__.__name__}: {result}")
```

```
DarcysLaw: 1250.0
MaterialBalanceOil: 1000000.0
VogelsIPR: 511.1111111111111
ArpsDecline: 666.6666666666666
STOIIPVolumetric: 4666666.666666667
GorRelationship: 375.0
```

ASSIGNMENT _2

```
def multiplication_table(x):
for t in range(1, 21):
    print(f'{x}×{t} = {x*t}')
num = int(input("Enter a number "))
multiplication_table(num)
```

```
Enter a number 5
5×1 = 5
5×2 = 10
5×3 = 15
5×4 = 20
5×5 = 25
5×6 = 30
5×7 = 35
5×8 = 40
5×9 = 45
5×10 = 50
5×11 = 55
5×12 = 60
5×13 = 65
5×14 = 70
5×15 = 75
5×16 = 80
5×17 = 85
5×18 = 90
5×19 = 95
5×20 = 100
```

```
import random

def get_player_guess():
    """
    Asks the player to enter a guess and handles potential errors.
    Returns the player's guess as an integer.
    """
    while True:
        try:
            guess = int(input("Guess a number between 1 and 10: "))
            return guess
        except ValueError:
            print("Invalid input! Please enter a whole number.")

def check_guess(secret_number, guess):
    """
    Compares the player's guess to the secret number.
    Returns True if the guess is correct, otherwise returns False.
    """
    if guess < secret_number:
        print("Too low! Try again.")
        return False
    elif guess > secret_number:
        print("Too high! Try again.")
        return False
    else:
        print(f"You got it! The number was {secret_number}.")
        return True
```

```
def play_game():
    """
    The main function to run the guessing game.
    """
    secret_number = random.randint(1, 10)
    max_attempts = 5
    attempts = 0
    guessed_correctly = False

    print("Welcome to the Guessing Game!")
    print(f"You have {max_attempts} attempts to guess the number.")

    while attempts < max_attempts and not guessed_correctly:
        attempts += 1
        print(f"\nAttempt {attempts}/{max_attempts}")
        player_guess = get_player_guess()
        guessed_correctly = check_guess(secret_number, player_guess)

    if not guessed_correctly:
        print(f"\nGame over! You ran out of attempts. The number was {secret_number}.")

# Start the game
play_game()
```

```
Welcome to the Guessing Game!
You have 5 attempts to guess the number.

Attempt 1/5
Guess a number between 1 and 10: 5
Too low! Try again.

Attempt 2/5
Guess a number between 1 and 10: 7
You got it! The number was 7.
```

ASSIGNMENT_3

```
def task1(s):
  """
  Replaces all uppercase letters in a string with their lowercase equivalents.

  Args:
    s: The input string.

  Returns:
    The modified string.
  """
  return s.lower()

# Examples:
print(f"Input: 'Hello', Output: '{task1('Hello')}'")
print(f"Input: 'hERE', Output: '{task1('hERE')}'")
print(f"Input: 'LOVELY', Output: '{task1('LOVELY')}'")
```

```
Input: 'Hello', Output: 'hello'
Input: 'hERE', Output: 'here'
```

```
    Input: 'LOVELY', Output: 'lovely'


def task2(s):
  """
  Swaps the case of all letters in a string.

  Args:
    s: The input string.

  Returns:
    The case-swapped string.
  """
  return s.swapcase()

# Example:
print(f"Input: 'HeLLo WoRLd', Output: '{task2('HeLLo WoRLd')}'")
```

```
Input: 'HeLLo WoRLd', Output: 'hEllO wOrlD'
```

```
def task3(s):
  """
  Removes all uppercase letters from a string.

  Args:
    s: The input string.

  Returns:
    The string with all uppercase letters removed.
  """
  modified_string = ""
  for char in s:
    if not char.isupper():
      modified_string += char
  return modified_string

# Example:
print(f"Input: 'HelloWorld', Output: '{task3('HelloWorld')}'")
```

```
Input: 'HelloWorld', Output: 'elloorld'
```

```
def task4(s):
  """
  Counts the number of uppercase and lowercase letters in a string.

  Args:
    s: The input string.

  Returns:
    A tuple containing the count of uppercase and lowercase letters.
  """
  uppercase_count = 0
  lowercase_count = 0
  for char in s:
    if char.isupper():
      uppercase_count += 1
    elif char.islower():
      lowercase_count += 1
```

```
  return (uppercase_count, lowercase_count)

# Example:
upper, lower = task4("EnginEEr")
print(f"Input: 'EnginEEr', Output: Uppercase: {upper}, Lowercase: {lower}")
```

```
Input: 'EnginEEr', Output: Uppercase: 3, Lowercase: 5
```

```
def task5(s):
  """
  Removes all non-alphabetic characters from a string.

  Args:
    s: The input string.

  Returns:
    The string containing only alphabetic characters.
  """
  modified_string = ""
  for char in s:
    if char.isalpha():
      modified_string += char
  return modified_string

# Example:
print(f"Input: 'Data-Driven@2025!', Output: '{task5('Data-Driven@2025!')}'")
```

```
Input: 'Data-Driven@2025!', Output: 'DataDriven'
```

```
import math

def task6(a, b, c):
  """
  Calculates the area of a triangle using Heron's formula.

  Args:
    a: Length of the first side.
    b: Length of the second side.
    c: Length of the third side.

  Returns:
    The area of the triangle.
  """
  # Calculate the semi-perimeter
  s = (a + b + c) / 2

  # Calculate the area using Heron's formula
  area = math.sqrt(s * (s - a) * (s - b) * (s - c))
  return area

# Example:
print(f"Input: a=3, b=4, c=5, Output: {task6(3, 4, 5):.1f}")
```

```
Input: a=3, b=4, c=5, Output: 6.0
```

```
def task7(names):
  """
```

```
  Formats and prints a list of names in a neat table.

  Args:
    names: A list of names (strings).
  """
  print("Name:".ljust(20) + "Length:".rjust(10))
  print("-" * 30)
  for name in names:
    length = len(name)
    print(name.ljust(20) + str(length).rjust(10))

# Example list of names:
name_list = ["Alice", "Bob", "Charlie", "David"]
task7(name_list)
```

```
Name:                  Length:
------------------------------
Alice                       5
Bob                         3
Charlie                     7
David                       5
```

```
import string

def task8(s):
  """
  Cleans a string by removing whitespace, punctuation, and spaces.

  Args:
    s: The input string.

  Returns:
    The cleaned string.
  """
  # i. Remove leading/trailing whitespace
  s = s.strip()

  # ii. Replace all punctuation with an empty string
  for punc in string.punctuation:
    s = s.replace(punc, "")

  # iii. Remove all spaces
  s = s.replace(" ", "")

  return s

# Example:
print(f"Input: ' Hello, World! ', Output: '{task8(' Hello, World! ')}'")
```

```
Input: ' Hello, World! ', Output: 'HelloWorld'
```