```
#multiplication table
def multiplication_table(n):
  for i in range(1, 21):
      print(f"{n}×{i} = {n*i}")
num = int(input("Enter a number "))
multiplication_table(num)
```

```
Enter a number 10
10×1 = 10
10×2 = 20
10×3 = 30
10×4 = 40
10×5 = 50
10×6 = 60
10×7 = 70
10×8 = 80
10×9 = 90
10×10 = 100
10×11 = 110
10×12 = 120
10×13 = 130
10×14 = 140
10×15 = 150
10×16 = 160
10×17 = 170
10×18 = 180
10×19 = 190
10×20 = 200
```

```
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 20: "))
            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 20: 10
Too high! Try again.

Attempt 2/5
Guess a number between 1 and 20: 15
Too high! Try again.

Attempt 3/5
Guess a number between 1 and 20: 17
Too high! Try again.

Attempt 4/5
Guess a number between 1 and 20: 13
Too high! Try again.

Attempt 5/5
Guess a number between 1 and 20: 2
Too high! Try again.

Game over! You ran out of attempts. The number was 1.
```

Assignment 3

```
def to_lowercase(s):
    return s.lower()

# Example
print(to_lowercase("Hello"))   # Output: "hello"
print(to_lowercase("LOVELY"))  # Output: "lovely"
```

```
hello
lovely
```

```
def swap_case(s):
    return s.swapcase()

# Example
print(swap_case("HeLLo WoRLd"))  # Output: "hEllO wOrlD"
```

```
hEllO wOrlD
```

```
def remove_uppercase(s):
    return ''.join(c for c in s if not c.isupper())

# Example
print(remove_uppercase("HelloWorld"))  # Output: "elloorld"
```

```
elloorld
```

```
def count_case(s):
    upper = sum(1 for c in s if c.isupper())
    lower = sum(1 for c in s if c.islower())
    return upper, lower

# Example
u, l = count_case("EngiNEEr")
print(f"Uppercase: {u}, Lowercase: {l}")  # Output: Uppercase: 4, Lowercase: 4
```

```
Uppercase: 4, Lowercase: 4
```

```
def remove_non_letters(s):
    return ''.join(c for c in s if c.isalpha())

# Example
print(remove_non_letters("Data-Driven@2025!"))  # Output: "DataDriven"
```

```
DataDriven
```

```
import math
```

```
def triangle_area(a, b, c):
    s = (a + b + c) / 2
    area = math.sqrt(s * (s - a) * (s - b) * (s - c))
    return area

# Example
print(triangle_area(3, 4, 5))  # Output: 6.0
```

```
6.0
```

```
import math

def triangle_area(a, b, c):
    s = (a + b + c) / 2
    area = math.sqrt(s * (s - a) * (s - b) * (s - c))
    return area

# Example
print(triangle_area(3, 4, 5))  # Output: 6.0
```

```
6.0
```

```
import string

def clean_string(s):
    s = s.strip()
    s = ''.join(c for c in s if c not in string.punctuation)
    s = s.replace(" ", "")
    return s

# Example
print(clean_string("   Hello, World!    "))  # Output: "HelloWorld"
```

```
HelloWorld
```

Assignment 1

```
# Create variables of different types
my_int = 10
my_float = 3.14
my_string = "Hello, Python!"
my_bool = True

# Print each variable with its type
print(my_int, type(my_int))
print(my_float, type(my_float))
print(my_string, type(my_string))
print(my_bool, type(my_bool))
```

```
10 <class 'int'>
3.14 <class 'float'>
Hello, Python! <class 'str'>
True <class 'bool'>
```

```
# a. Convert float 19.99 to integer
num_float = 19.99
num_int = int(num_float)
print(num_int, type(num_int))

# b. Convert integer 50 to string
num = 50
num_str = str(num)
print(num_str, type(num_str))

# c. Convert string "50" to float
str_num = "50"
float_num = float(str_num)
print(float_num, type(float_num))
```

```
19 <class 'int'>
50 <class 'str'>
50.0 <class 'float'>
```

```
# Ask for favorite word and number of repetitions
word = input("Enter your favorite word: ")
times = int(input("How many times should I repeat it? "))

# Print the word that many times, separated by spaces
print((word + " ") * times)
```

```
# Given code
age = 20

# a. Fix the error
print("You are " + str(age) + " years old.")

# b. Why does the error occur?
# Because you cannot concatenate a string and an integer directly.
# You must convert the integer to a string using str().
```

```
You are 20 years old.
```

```
#Assignment 4
import math

# Base class for any drilling formula
class DrillingFormula:
    def calculate(self):
        raise NotImplementedError("This method should be overridden in subclasses")
```

```
class HydrostaticPressure(DrillingFormula):
    def __init__(self, mud_weight, tvd):
        self.mw = mud_weight
        self.tvd = tvd

    def calculate(self):
        return 0.052 * self.mw * self.tvd


class AnnularVelocity(DrillingFormula):
    def __init__(self, flow_rate, casing_id, pipe_od):
        self.q = flow_rate
        self.id = casing_id
        self.od = pipe_od

    def calculate(self):
        if self.id <= self.od:
            raise ValueError("Casing ID must be larger than pipe OD")
        return (24.5 * self.q) / (self.id**2 - self.od**2)


class PumpOutput(DrillingFormula):
    def __init__(self, liner_diameter, stroke_length):
        self.d = liner_diameter
        self.s = stroke_length

    def calculate(self):
        return (math.pi * self.d**2 * self.s) / 4


class PumpFlowRate(DrillingFormula):
    def __init__(self, pump_output, strokes_per_min):
        self.po = pump_output
        self.n = strokes_per_min

    def calculate(self):
        return self.po * self.n


class EquivalentCirculatingDensity(DrillingFormula):
    def __init__(self, mud_weight, circulating_pressure, tvd):
        self.mw = mud_weight
        self.pc = circulating_pressure
        self.tvd = tvd

    def calculate(self):
        if self.tvd == 0:
            raise ZeroDivisionError("TVD cannot be zero")
        return self.mw + (self.pc / (0.052 * self.tvd))


class PressureLoss(DrillingFormula):
    def __init__(self, friction_factor, length, diameter, density, velocity):
        self.f = friction_factor
        self.l = length
        self.d = diameter
        self.rho = density
        self.v = velocity

    def calculate(self):
        if self.d == 0:
            raise ZeroDivisionError("Diameter cannot be zero")
        return self.f * (self.l / self.d) * (self.rho * self.v**2 / 2)


# --- Demonstrate Polymorphism ---
def calculate_formula(formula_obj):
    # This function works with ANY subclass of DrillingFormula
    try:
        result = formula_obj.calculate()
        print(f"{formula_obj.__class__.__name__}: {result:.3f}")
    except Exception as e:
        print(f"Error in {formula_obj.__class__.__name__}: {e}")


# --- Main Program (Test Objects) ---
if __name__ == "__main__":
    # Create objects
    f1 = HydrostaticPressure(mud_weight=12.5, tvd=8000)
    f2 = AnnularVelocity(flow_rate=500, casing_id=9.0, pipe_od=5.0)
    f3 = PumpOutput(liner_diameter=6.0, stroke_length=12.0)
    f4 = PumpFlowRate(pump_output=5.5, strokes_per_min=120)
    f5 = EquivalentCirculatingDensity(mud_weight=12.5, circulating_pressure=400, tvd=8000)
    f6 = PressureLoss(friction_factor=0.02, length=5000, diameter=5, density=9.5, velocity=

    # Run all using polymorphic function
    formulas = [f1, f2, f3, f4, f5, f6]
    for f in formulas:
        calculate_formula(f)
```

```
HydrostaticPressure: 5200.000
AnnularVelocity: 218.750
PumpOutput: 339.292
PumpFlowRate: 660.000
EquivalentCirculatingDensity: 13.462
PressureLoss: 9500.000
```