

# LAB ASSIGNMENT 3.5

Subject : AI Assistant Coding

Hall Ticket No : 2303A52465

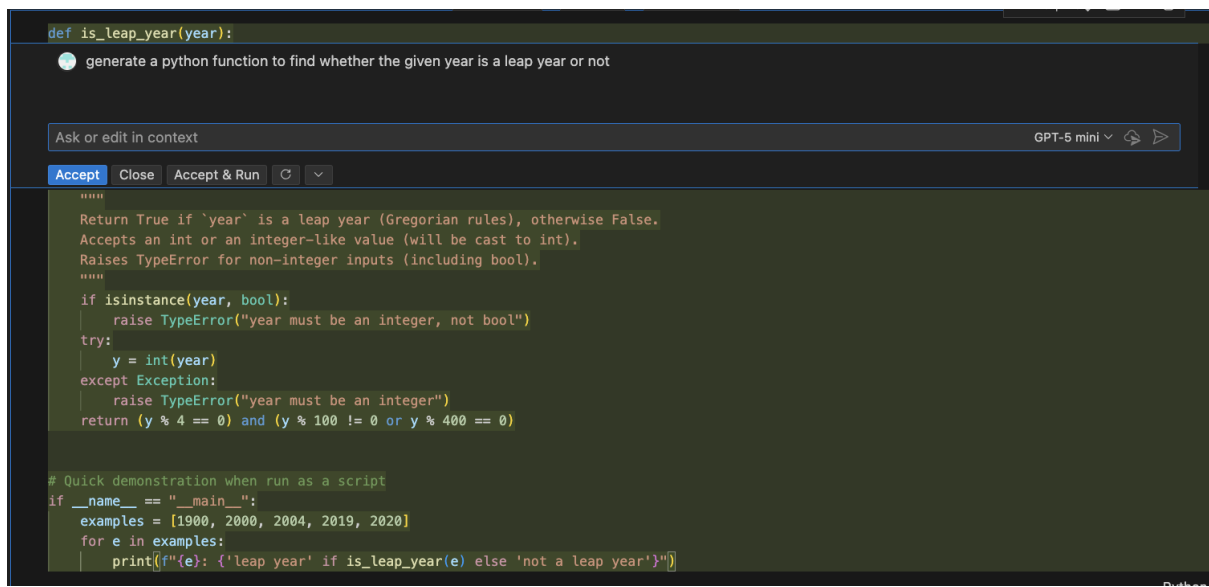
Name : M.Avinash

Batch : 50

Date : 23/1/26

## Task -1:

**Prompt:** generate a python function to find whether the given year is a leap year or not.

A screenshot of a code editor interface. At the top, a prompt says "generate a python function to find whether the given year is a leap year or not". Below the prompt is a text input field with the placeholder "Ask or edit in context". To the right of the input field are buttons for "Accept", "Close", "Accept & Run", and a dropdown menu. The main area of the editor displays a Python function named "is\_leap\_year". The function includes docstrings, type hints, and a try-except block to handle non-integer inputs. It also includes a quick demonstration when run as a script, showing examples for the years 1900, 2000, 2004, 2019, and 2020.

```
def is_leap_year(year):  
    """  
    Return True if 'year' is a leap year (Gregorian rules), otherwise False.  
    Accepts an int or an integer-like value (will be cast to int).  
    Raises TypeError for non-integer inputs (including bool).  
    """  
    if isinstance(year, bool):  
        raise TypeError("year must be an integer, not bool")  
    try:  
        y = int(year)  
    except Exception:  
        raise TypeError("year must be an integer")  
    return (y % 4 == 0) and (y % 100 != 0 or y % 400 == 0)  
  
# Quick demonstration when run as a script  
if __name__ == "__main__":  
    examples = [1900, 2000, 2004, 2019, 2020]  
    for e in examples:  
        print(f"{e}: {'leap year' if is_leap_year(e) else 'not a leap year'}")
```

## Output :

```
1900: not a leap year  
2000: leap year  
2004: leap year  
2019: not a leap year  
2020: leap year
```

## Task 2:

**Prompt:** generate a python code with one example to find the GCD of two numbers.

```
def gcd(a, b):
    """Return the greatest common divisor of two integers (Euclidean algorithm)."""
    a, b = int(a), int(b)
    a, b = abs(a), abs(b)
    if a == 0:
        return b
    if b == 0:
        return a
    while b:
        a, b = b, a % b
    return a

# Example using existing notebook variable `e` (2020) and another number
x = int(input())
y = int(input())
print(f"GCD({x}, {y}) = {gcd(x, y)}")
```

**Output:**

✓ 7.3s Python

GCD(12, 18) = 6

### Task 3:

**Prompt :** generate a python code for multiple examples to find LCM Calculations.

```
def lcm(a, b):
    """Return least common multiple of two integers. If either is 0, return 0."""
    a, b = int(a), int(b)
    if a == 0 or b == 0:
        return 0
    return abs(a * b) // gcd(a, b)
```

```
def lcm_list(nums):
    """Return LCM of a list of integers (returns 1 for empty list)."""
    nums = [int(n) for n in nums]
    if not nums:
        return 1
    result = nums[0]
    for n in nums[1:]:
        result = lcm(result, n)
    return result
```

```
# Examples using existing notebook variables: x, y, e, examples
print(f"LCM({x}, {y}) = {lcm(x, y)}")

for n in examples:
    print(f"LCM({e}, {n}) = {lcm(e, n)}")

print(f"LCM of the list {examples} = {lcm_list(examples)}")

# Pairwise LCMs for adjacent items in examples
print("Pairwise LCMs of adjacent items in examples:")
for a, b in zip(examples, examples[1:]):
    print(f"LCM({a}, {b}) = {lcm(a, b)}")
```

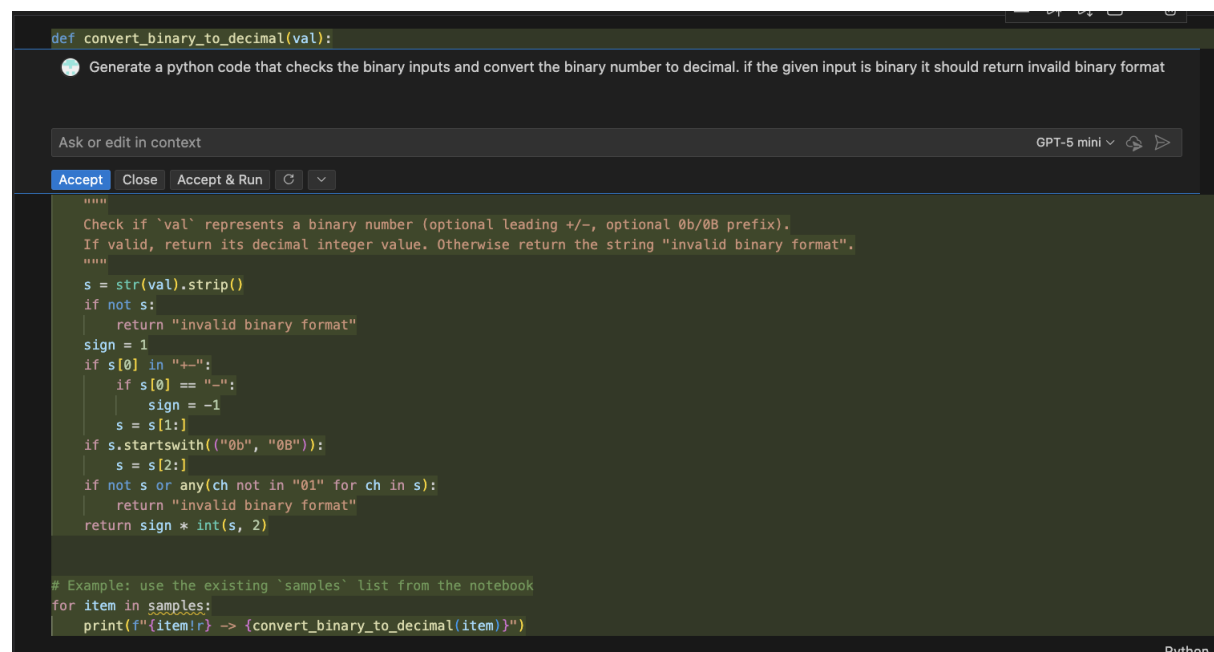
Python

## Output :

```
LCM(12, 18) = 36
LCM(2020, 1900) = 191900
LCM(2020, 2000) = 202000
LCM(2020, 2004) = 1012020
LCM(2020, 2019) = 4078380
LCM(2020, 2020) = 2020
LCM of the list [1900, 2000, 2004, 2019, 2020] = 1294069974000
Pairwise LCMs of adjacent items in examples:
LCM(1900, 2000) = 38000
LCM(2000, 2004) = 1002000
LCM(2004, 2019) = 1348692
LCM(2019, 2020) = 4078380
```

## Task 4 :

**Prompt :** Generate a python code that checks the binary inputs and convert the binary number to decimal. if the given input is binary it should return invalid binary format.



```
def convert_binary_to_decimal(val):
    """
    Generate a python code that checks the binary inputs and convert the binary number to decimal. if the given input is binary it should return invalid binary format
    """
    Ask or edit in context
    GPT-5 mini
    Accept Close Accept & Run
    """
    Check if 'val' represents a binary number (optional leading +/-, optional 0b/0B prefix).
    If valid, return its decimal integer value. Otherwise return the string "invalid binary format".
    """
    s = str(val).strip()
    if not s:
        return "invalid binary format"
    sign = 1
    if s[0] in "+-":
        if s[0] == "-":
            sign = -1
        s = s[1:]
    if s.startswith(("0b", "0B")):
        s = s[2:]
    if not s or any(ch not in "01" for ch in s):
        return "invalid binary format"
    return sign * int(s, 2)

# Example: use the existing 'samples' list from the notebook
for item in samples:
    print(f"{item!r} -> {convert_binary_to_decimal(item)}")
```

## Output :

```
'1010' -> 10
'0b110' -> 6
'-111' -> -7
' 00101 ' -> 5
'+0B1001' -> 9
'1234' -> invalid binary format
```

## Task 5 :

**Prompt :** generate Python function with examples that converts decimal number to binary.

```
def decimal_to_binary(value, prefix=True, width=None):  
    """  
    generate Python function with examples that converts decimal number to binary  
    """  
    Ask or edit in context  
    GPT-5 mini  
    Accept Close Accept & Run  
    """  
    Convert an integer-like value to its binary representation.  
    - value: int or integer-like (will be cast to int). Raises TypeError for non-integer inputs (including bool).  
    - prefix: if True, include '0b' after any sign.  
    - width: optional minimum number of binary digits (zero-padded on the left).  
    Returns a string like '0b1010', '-0b101', or '1010' when prefix=False.  
    """  
    if isinstance(value, bool):  
        raise TypeError("value must be an integer, not bool")  
    try:  
        v = int(value)  
    except Exception:  
        raise TypeError("value must be an integer")  
    sign = '-' if v < 0 else ''  
    mag = abs(v)  
    b = format(mag, 'b')  
    if width is not None:  
        try:  
            w = int(width)  
        except Exception:  
            raise TypeError("width must be an integer")  
        if w < 0:  
            raise ValueError("width must be non-negative")  
        if len(b) < w:  
            b = b.rjust(w, '0')  
    return f"{sign}{b}" if prefix else f"{b}"  
  
# Examples using existing notebook variables  
print(f"a={a} -> {decimal_to_binary(a)}") # uses a (2019)  
print(f"e={e} -> {decimal_to_binary(e, prefix=False)}") # no 0b prefix  
print(f"n={n} -> {decimal_to_binary(n)}") # uses n (2020)  
print(f"x={x}, y={y} with width=8 -> {decimal_to_binary(x, width=8)}, {decimal_to_binary(y, width=8)}")  
print(f"string b='{b}' -> {decimal_to_binary(b)}") # accepts integer-like string  
  
# Convert all items in examples list  
print("examples -> binaries:")  
for num in examples:  
    print(f"{num} -> {decimal_to_binary(num)}")  
  
# Show handling of negative number  
print(f"-5 -> {decimal_to_binary(-5)}")
```

## Output:

```
✓ 0.0s Python  
  
a=2019 -> 0b11111100011  
e=2020 -> 11111100100  
n=2020 -> 0b11111100100  
x=12, y=18 with width=8 -> 0b00001100, 0b00010010  
string b='1234' -> 0b10011010010  
examples -> binaries:  
1900 -> 0b11101101100  
2000 -> 0b11111010000  
2004 -> 0b11111010100  
2019 -> 0b11111100011  
2020 -> 0b11111100100  
-5 -> -0b101
```

## Task 6 :

**Prompt :** generate a python function that checks whether the given number is a Harshad number ,and print the number as Harshad number or not.

```
def is_harshad(value):  
    """  
    generate a python function that checks whether the given number is a Harshad number ,and print the number as Harshad number or not  
    """  
    Ask or edit in context GPT-5 mini  
    Accept Close Accept & Run  
    """  
    Return True if integer-like value is a Harshad (Niven) number:  
    divisible by the sum of its decimal digits.  
    Raises TypeError for non-integer inputs (including bool).  
    """  
    if isinstance(value, bool):  
        raise TypeError("value must be an integer, not bool")  
    try:  
        n = int(value)  
    except Exception:  
        raise TypeError("value must be an integer")  
    m = abs(n)  
    digit_sum = sum(int(d) for d in str(m))  
    if digit_sum == 0:  
        return False  
    return (m % digit_sum) == 0  
  
def print_harshad(value):  
    """Print whether value is a Harshad number."""  
    if isinstance(value, bool):  
        raise TypeError("value must be an integer, not bool")  
    try:  
        n = int(value)  
    except Exception:  
        raise TypeError("value must be an integer")  
    if is_harshad(n):  
        print(f"{n} is a Harshad number")  
    else:  
        print(f"{n} is not a Harshad number")  
  
# Example using existing notebook variable `num`  
print_harshad(num)
```

**Output :**

```
✓ 0.0s Python  
  
18 is a Harshad number  
21 is a Harshad number  
19 is not a Harshad number
```