Python Programming - Conditional Statement

1. **Even or Odd:**

```
num = int(input("Enter a number: "))
if num % 2 == 0:
    print("Even")
else:
    print("Odd")
```

Checks if a number is even or odd.

2. **Positive, Negative, or Zero:**

```
num = float(input("Enter a number: "))
if num > 0:
    print("Positive")
elif num < 0:
    print("Negative")
else:
    print("Zero")
```

Determines whether a number is positive, negative, or zero.

3. **Leap Year:**

```
year = int(input("Enter a year: "))
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print("Leap Year")
else:
    print("Not a Leap Year")
```

Identifies if a given year is a leap year or not.

4. **Vowel or Consonant:**

```
char = input("Enter a character: ")
if char.lower() in 'aeiou':
    print("Vowel")
else:
    print("Consonant")
```

   Determines if a character is a vowel or consonant.

5. **Maximum of Three Numbers:**

```
a, b, c = map(int, input("Enter three numbers: ").split())
max_num = max(a, b, c)
print("Maximum:", max_num)
```

   Finds the maximum among three given numbers.

6. **Grade Classification:**

```
marks = float(input("Enter marks: "))
if marks >= 90:
    print("A")
elif 80 <= marks < 90:
    print("B")
elif 70 <= marks < 80:
    print("C")
else:
    print("Fail")
```

   Classifies grades based on marks.

7. **Check for Palindrome:**

```
word = input("Enter a word: ")
if word == word[::-1]:
    print("Palindrome")
else:
    print("Not a Palindrome")
```

   Determines if a word is a palindrome.

8. **Check for Prime Number:**

```
num = int(input("Enter a number: "))
if num > 1:
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            print("Not Prime")
            break
    else:
        print("Prime")
else:
    print("Not Prime")
```

Checks if a number is prime.

9. **Check for a Divisible Number:**

```
num = int(input("Enter a number: "))
if num % 7 == 0 and num % 5 == 0:
    print("Divisible by 7 and 5")
else:
    print("Not Divisible")
```

Determines if a number is divisible by both 7 and 5.

10. **Check for a Perfect Square:**

```
num = int(input("Enter a number: "))
if num > 0 and int(num**0.5)**2 == num:
    print("Perfect Square")
else:
    print("Not a Perfect Square")
```

Identifies if a number is a perfect square.

11. **Check for a Valid Triangle:**

```
a, b, c = map(int, input("Enter three sides of a triangle: ").split())
if a + b > c and a + c > b and b + c > a:
    print("Valid Triangle")
else:
    print("Invalid Triangle")
```

Checks if three given sides can form a valid triangle.

12. **Check for Divisibility by 3 and 5:**

```
num = int(input("Enter a number: "))
if num % 3 == 0 and num % 5 == 0:
    print("Divisible by both 3 and 5")
```

```
else:
    print("Not Divisible by both 3 and 5")
```

Determines if a number is divisible by both 3 and 5.

13. **Check for a Power of Two:**

```
num = int(input("Enter a number: "))
if num > 0 and (num & (num - 1)) == 0:
    print("Power of Two")
else:
    print("Not a Power of Two")
```

Determines if a number is a power of two.

14. **Check for a Valid Password:**

```
password = input("Enter a password: ")
if len(password) >= 8 and any(c.isdigit() for c in password) and any(c.isalpha() for c in
    print("Valid Password")
else:
    print("Invalid Password")
```

Validates a password based on length, digits, and letters.

15. **Check for a Valid Email Address:**

```
email = input("Enter an email address: ")
if "@" in email and "." in email:
    print("Valid Email Address")
else:
    print("Invalid Email Address")
```

Validates an email address.

16. **Check for a Positive Number:**

```
num = float(input("Enter a number: "))
if num > 0:
    print("Positive Number")
else:
    print("Non-Positive Number")
```

Checks if a number is positive.

17. **Check for Equality of Three Numbers:**

```
a, b, c = map(int, input("Enter three numbers: ").split())
if a == b == c:
    print("All numbers are equal")
else:
    print("Not all numbers are equal")
```

Checks if three numbers are equal.

18. **Check for Anagram Strings:**

```
str1, str2 = input("Enter two strings: ").split()
if sorted(str1) == sorted(str2):
    print("Anagram")
else:
    print("Not an Anagram")
```

Checks if two strings are anagrams.

19. **Check for a Valid Date:**

```
day, month, year = map(int, input("Enter date (DD MM YYYY): ").split())
if 1 <= day <= 31 and 1 <= month <= 12 and 1900 <= year <= 2100:
    print("Valid Date")
else:
    print("Invalid Date")
```

Validates a given date.

20. **Check for a Perfect Number:**

```
num = int(input("Enter a number: "))
sum_of_divisors = sum(i for i in range(1, num) if num % i == 0)
if sum_of_divisors == num:
    print("Perfect Number")
else:
    print("Not a Perfect Number")
```

Checks if a number is a perfect number.

21. **Check for a Valid Credit Card Number:**

```
card_number = input("Enter a credit card number: ")
if card_number.isdigit() and len(card_number) == 16:
    print("Valid Credit Card Number")
```

```
else:
    print("Invalid Credit Card Number")
```

Validates a credit card number based on length and digits.

22. **Check for a Triangle Type:**

```
a, b, c = map(int, input("Enter three sides of a triangle: ").split())
if a == b == c:
    print("Equilateral Triangle")
elif a == b or b == c or a == c:
    print("Isosceles Triangle")
else:
    print("Scalene Triangle")
```

Determines the type of triangle based on side lengths.

23. **Check for a Perfect Cube:**

```
num = int(input("Enter a number: "))
if num > 0 and int(num**(1/3))**3 == num:
    print("Perfect Cube")
else:
    print("Not a Perfect Cube")
```

Identifies if a number is a perfect cube.

24. **Check for a Pangram Sentence:**

```
sentence = input("Enter a sentence: ").lower()
alphabet_set = set('abcdefghijklmnopqrstuvwxyz')
if set(sentence) >= alphabet_set:
    print("Pangram")
else:
    print("Not a Pangram")
```

Determines if a sentence is a pangram.

25. **Check for a Quadrilateral Type:**

```
angles = list(map(int, input("Enter four angles of a quadrilateral: ").split()))
if sum(angles) == 360:
    if all(angle > 0 for angle in angles):
        if any(angle == 90 for angle in angles):
            print("Rectangle or Square")
        else:
            print("Quadrilateral")
    else:
        print("Invalid Angles")
else:
    print("Not a Quadrilateral")
```

Identifies the type of quadrilateral based on angle measures.

26. **Check for a Fibonacci Number:**

```
num = int(input("Enter a number: "))
a, b = 0, 1
while b < num:
    a, b = b, a + b
if b == num:
    print("Fibonacci Number")
else:
    print("Not a Fibonacci Number")
```

Checks if a number is a Fibonacci number.

27. **Check for a Valid IPv4 Address:**

```
ip_address = input("Enter an IPv4 address: ")
octets = ip_address.split('.')
if len(octets) == 4 and all(octet.isdigit() and 0 <= int(octet) <= 255 for octet in octets
    print("Valid IPv4 Address")
else:
    print("Invalid IPv4 Address")
```

Validates an IPv4 address.

28. **Check for a Valid URL:**

```
url = input("Enter a URL: ")
if url.startswith("http://") or url.startswith("https://"):
    print("Valid URL")
else:
    print("Invalid URL")
```

Validates a URL based on the protocol.

29. **Check for a Perfect Number in a Range:**

```
lower, upper = map(int, input("Enter a range (lower upper): ").split())
for num in range(lower, upper + 1):
    sum_of_divisors = sum(i for i in range(1, num) if num % i == 0)
    if sum_of_divisors == num:
        print(num, "is a Perfect Number")
```

Finds and prints perfect numbers in a given range.

30. **Check for a Valid ISBN-13:**

```
isbn = input("Enter an ISBN-13: ")
if len(isbn) == 13 and isbn.isdigit():
    checksum = sum(int(isbn[i]) * (3 if i % 2 == 0 else 1) for i in range(12))
    if (10 - (checksum % 10)) % 10 == int(isbn[12]):
        print("Valid ISBN-13")
    else:
        print("Invalid ISBN-13")
else:
    print("Invalid ISBN-13 format")
```

Validates an ISBN-13 based on the checksum.

31. **Check for a Magic Number:**

```
num = int(input("Enter a number: "))
if num > 0 and num == sum(int(digit) for digit in str(num)):
    print("Magic Number")
else:
    print("Not a Magic Number")
```

Determines if a number is a magic number.

32. **Check for a Hamming Number:**

```
num = int(input("Enter a number: "))
while num % 2 == 0:
    num //= 2
while num % 3 == 0:
    num //= 3
while num % 5 == 0:
    num //= 5
if num == 1:
    print("Hamming Number")
else:
    print("Not a Hamming Number")
```

Identifies if a number is a Hamming number.

33. **Check for a Strong Number:**

```python
num = int(input("Enter a number: "))
factorial_sum = sum(int(fact_digit) for fact_digit in str(num))
if factorial_sum == num:
    print("Strong Number")
else:
    print("Not a Strong Number")
```

Checks if a number is a strong number.

34. **Check for a Harshad Number:**

```python
num = int(input("Enter a number: "))
if num % sum(int(digit) for digit in str(num)) == 0:
    print("Harshad Number")
else:
    print("Not a Harshad Number")
```

Identifies if a number is a Harshad number.

35. **Check for a Duck Number:**

```python
num = input("Enter a number: ")
if '0' in num and num[0] != '0':
    print("Duck Number")
else:
    print("Not a Duck Number")
```

Determines if a number is a duck number.

36. **Check for a Happy Number:**

```python
num = int(input("Enter a number: "))
seen = set()
while num != 1 and num not in seen:
    seen.add(num)
    num = sum(int(digit)**2 for digit in str(num))
if num == 1:
    print("Happy Number")
else:
    print("Not a Happy Number")
```

Checks if a number is a happy number.

37. **Check for a Neon Number:**

```
num = int(input("Enter a number: "))
square_sum = sum(int(digit)**2 for digit in str(num**2))
if square_sum == num:
    print("Neon Number")
else:
    print("Not a Neon Number")
```

Identifies if a number is a neon number.

38. **Check for a Pronic Number:**

```
num = int(input("Enter a number: "))
for i in range(num):
    if i * (i + 1) == num:
        print("Pronic Number")
        break
else:
    print("Not a Pronic Number")
```

Determines if a number is a pronic number.

39. **Check for a Spy Number:**

```
num = int(input("Enter a number: "))
digits = [int(digit) for digit in str(num)]
if sum(digits) == math.prod(digits):
    print("Spy Number")
else:
    print("Not a Spy Number")
```

Checks if a number is a spy number.

40. **Check for a Happy Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
seen = set()
while num != 1 and num not in seen and is_prime(num):
    seen.add(num)
    num = sum(int(digit)**2 for digit in str(num))
if num == 1 and is_prime(num):
    print("Happy Prime Number")
else:
    print("Not a Happy Prime Number")
```

Checks if a number is a happy prime number.

41. **Check for a Strong Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def factorial_sum(n):
    return sum(math.factorial(int(digit)) for digit in str(n))
if is_prime(num) and is_prime(factorial_sum(num)):
    print("Strong Prime Number")
else:
    print("Not a Strong Prime Number")
```

Checks if a number is a strong prime number.

42. **Check for an Emirp Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
reversed_num = int(str(num)[::-1])
if is_prime(num) and is_prime(reversed_num) and num != reversed_num:
    print("Emirp Number")
else:
    print("Not an Emirp Number")
```

Identifies if a number is an emirp number.

43. **Check for a Fibonacci Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
```

```python
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_fibonacci(n):
    a, b = 0, 1
    while b < n:
        a, b = b, a + b
    return b == n
if is_prime(num) and is_fibonacci(num):
    print("Fibonacci Prime Number")
else:
    print("Not a Fibonacci Prime Number")
```

Checks if a number is a Fibonacci prime number.

44. **Check for a Happy Square Number:**

```python
num = int(input("Enter a number: "))
def is_happy(n):
    seen = set()
    while n != 1 and n not in seen:
        seen.add(n)
        n = sum(int(digit)**2 for digit in str(n))
    return n == 1
if is_happy(num) and int(num**0.5)**2 == num:
    print("Happy Square Number")
else:
    print("Not a Happy Square Number")
```

Determines if a number is a happy square number.

45. **Check for an Automorphic Number:**

```python
num = int(input("Enter a number: "))
square = num**2
if str(square).endswith(str(num)):
    print("Automorphic Number")
else:
    print("Not an Automorphic Number")
```

Checks if a number is an automorphic number.

46. **Check for a Perfect Power Number:**

```python
num = int(input("Enter a number: "))
for base in range(2, int(num**0.5) + 1):
    power = 2
    while base**power <= num:
        if base**power == num:
            print("Perfect Power Number")
            break
        power += 1
else:
    print("Not a Perfect Power Number")
```

Identifies if a number is a perfect power number.

47. **Check for an Armstrong Number:**

```
num = int(input("Enter a number: "))
order = len(str(num))
armstrong_sum = sum(int(digit)**order for digit in str(num))
if armstrong_sum == num:
    print("Armstrong Number")
else:
    print("Not an Armstrong Number")
```

Determines if a number is an Armstrong number.

48. **Check for a Smith Number:**

```
num = int(input("Enter a number: "))
def prime_factors(n):
    factors = []
    for i in range(2, n + 1):
        while n % i == 0:
            factors.append(i)
            n //= i
    return factors
if sum(map(int, str(num))) == sum(map(lambda x: sum(map(int, str(x))), prime_factors(num))
    print("Smith Number")
else:
    print("Not a Smith Number")
```

Checks if a number is a Smith number.

49. **Check for a Vampire Number:**

```
num = int(input("Enter a number: "))
def is_vampire_number(n, fangs):
    product = fangs[0] * fangs[1]
    return sorted(str(n)) == sorted(str(product))
```

```python
digits = len(str(num))
for i in range(10**(digits//2 - 1), int(num**0.5) + 1):
    if num % i == 0:
        fang1, fang2 = i, num // i
        if is_vampire_number(num, [fang1, fang2]):
            print("Vampire Number")
            break
else:
    print("Not a Vampire Number")
```

Identifies if a number is a vampire number.

50. **Check for a Triangular Number:**

```python
num = int(input("Enter a number: "))
triangular_sum = 0
i = 1
while triangular_sum < num:
    triangular_sum += i
    i += 1
if triangular_sum == num:
    print("Triangular Number")
else:
    print("Not a Triangular Number")
```

Checks if a number is a triangular number.

51. **Check for an Achilles Number:**

```python
num = int(input("Enter a number: "))
def prime_factors(n):
    factors = []
    for i in range(2, n + 1):
        while n % i == 0:
            factors.append(i)
            n //= i
    return factors
def is_powerful(n):
    distinct_factors = set(prime_factors(n))
    for factor in distinct_factors:
        if prime_factors(n).count(factor) > 1:
            return True
    return False
if is_powerful(num) and not is_powerful(num**2):
    print("Achilles Number")
else:
    print("Not an Achilles Number")
```

Determines if a number is an Achilles number.

52. **Check for a Neon Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
square_sum = sum(int(digit)**2 for digit in str(num**2))
if is_prime(num) and square_sum == num:
    print("Neon Prime Number")
else:
    print("Not a Neon Prime Number")
```

Checks if a number is a neon prime number.

53. **Check for a Keith Number:**

```
num = int(input("Enter a number: "))
series = [int(digit) for digit in str(num)]
while series[-1] < num:
    series.append(sum(series[-len(series):]))
if num in series:
    print("Keith Number")
else:
    print("Not a Keith Number")
```

Identifies if a number is a Keith number.

54. **Check for a Centered Hexagonal Number:**

```
num = int(input("Enter a number: "))
hexagonal_num = 1 + 3 * (num - 1) * num
if hexagonal_num == num * (2 * num - 1):
    print("Centered Hexagonal Number")
else:
    print("Not a Centered Hexagonal Number")
```

Checks if a number is a centered hexagonal number.

55. **Check for a Happy Square Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_happy(n):
    seen = set()
    while n != 1 and n not in seen:
        seen.add(n)
        n = sum(int(digit)**2 for digit in str(n))
    return n == 1
square_sum = sum(int(digit)**2 for digit in str(num**2))
if is_prime(num) and is_happy(square_sum):
    print("Happy Square Prime Number")
else:
    print("Not a Happy Square Prime Number")
```

Checks if a number is a happy square prime number.

56. **Check for a Fibonacci Cube Number:**

```
num = int(input("Enter a number: "))
def is_fibonacci(n):
    a, b = 0, 1
    while b < n:
        a, b = b, a + b
    return b == n
cube_root = round(num**(1/3))
if is_fibonacci(num) and cube_root**3 == num:
    print("Fibonacci Cube Number")
else:
    print("Not a Fibonacci Cube Number")
```

Determines if a number is a Fibonacci cube number.

57. **Check for a Decagonal Number:**

```
num = int(input("Enter a number: "))
decagonal_num = num * (9 * num - 7)
if (24 * decagonal_num + 1)**0.5 % 6 == 5:
    print("Decagonal Number")
else:
    print("Not a Decagonal Number")
```

Checks if a number is a decagonal number.

58. **Check for a Unique Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
num_digits = set(str(num))
if is_prime(num) and len(num_digits) == len(str(num)):
    print("Unique Prime Number")
else:
    print("Not a Unique Prime Number")
```

Identifies if a number is a unique prime number.

59. **Check for a Regular Number:**

```
num = int(input("Enter a number: "))
def is_regular(n):
    for factor in [2, 3, 5]:
        while n % factor == 0:
            n //= factor
    return n == 1
if is_regular(num):
    print("Regular Number")
else:
    print("Not a Regular Number")
```

Checks if a number is a regular number.

60. **Check for a Harshad Cube Number:**

```
num = int(input("Enter a number: "))
def is_harshad(n):
    return n % sum(int(digit) for digit in str(n)) == 0
cube_root = round(num**(1/3))
if is_harshad(num) and cube_root**3 == num:
    print("Harshad Cube Number")
else:
    print("Not a Harshad Cube Number")
```

Determines if a number is a Harshad cube number.

61. **Check for a Perfect Digital Invariant Number:**

```
num = int(input("Enter a number: "))
digit_sum = sum(int(digit) for digit in str(num))
product_of_digits = math.prod(int(digit) for digit in str(num) if int(digit) != 0)
if digit_sum * product_of_digits == num:
    print("Perfect Digital Invariant Number")
else:
    print("Not a Perfect Digital Invariant Number")
```

Checks if a number is a perfect digital invariant number.

62. **Check for a Mersenne Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_mersenne_prime(p):
    return is_prime(2**p - 1)
if is_prime(num) and is_mersenne_prime(num):
    print("Mersenne Prime Number")
else:
    print("Not a Mersenne Prime Number")
```

Identifies if a number is a Mersenne prime number.

63. **Check for a Lucas Number:**

```
num = int(input("Enter a number: "))
def is_perfect_square(n):
    return int(n**0.5)**2 == n
lucas_test1 = 5 * (num**2) + 4
lucas_test2 = 5 * (num**2) - 4
if is_perfect_square(lucas_test1) or is_perfect_square(lucas_test2):
    print("Lucas Number")
else:
    print("Not a Lucas Number")
```

Checks if a number is a Lucas number.

64. **Check for a Regular Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_regular(n):
    for factor in [2, 3, 5]:
        while n % factor == 0:
            n //= factor
    return n == 1
if is_prime(num) and is_regular(num):
    print("Regular Prime Number")
else:
    print("Not a Regular Prime Number")
```

Identifies if a number is a regular prime number.

65. **Check for a Harshad Smith Number:**

```
num = int(input("Enter a number: "))
def is_harshad(n):
    return n % sum(int(digit) for digit in str(n)) == 0
def prime_factors(n):
    factors = []
    for i in range(2, n + 1):
        while n % i == 0:
            factors.append(i)
            n //= i
    return factors
if is_harshad(num) and sum(prime_factors(num)) == sum(int(digit) for digit in str(num)):
    print("Harshad Smith Number")
else:
    print("Not a Harshad Smith Number")
```

Checks if a number is a Harshad Smith number.

66. **Check for a Triangular Square Number:**

```
num = int(input("Enter a number: "))
triangular_test = 8 * num + 1
if math.isqrt(triangular_test)**2 == triangular_test:
    print("Triangular Square Number")
else:
    print("Not a Triangular Square Number")
```

Determines if a number is a triangular square number.

67. **Check for a Palindromic Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
if is_prime(num) and str(num) == str(num)[::-1]:
    print("Palindromic Prime Number")
else:
    print("Not a Palindromic Prime Number")
```

Checks if a number is a palindromic prime number.

68. **Check for a Sophisticated Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_sophisticated_prime(n):
    return is_prime(n) and is_prime(int(str(n)[::-1]))
if is_sophisticated_prime(num):
    print("Sophisticated Prime Number")
else:
    print("Not a Sophisticated Prime Number")
```

Identifies if a number is a sophisticated prime number.

69. **Check for a Harshad Cube Harshad Number:**

```
num = int(input("Enter a number: "))
def is_harshad(n):
    return n % sum(int(digit) for digit in str(n)) == 0
cube_root = round(num**(1/3))
if is_harshad(num) and is_harshad(cube_root):
    print("Harshad Cube Harshad Number")
else:
    print("Not a Harshad Cube Harshad Number")
```

Checks if a number is a Harshad Cube Harshad number.

70. **Check for a Harshad Smith Cube Harshad Number:**

```
num = int(input("Enter a number: "))
def is_harshad(n):
    return n % sum(int(digit) for digit in str(n)) == 0
def prime_factors(n):
    factors = []
    for i in range(2, n + 1):
        while n % i == 0:
            factors.append(i)
            n //= i
    return factors
cube_root = round(num**(1/3))
if is_harshad(num) and sum(prime_factors(num)) == sum(int(digit) for digit in str(num)) ar
    print("Harshad Smith Cube Harshad Number")
else:
    print("Not a Harshad Smith Cube Harshad Number")
```

Identifies if a number is a Harshad Smith Cube Harshad number.

71. **Check for a Repunit Number:**

```
num = int(input("Enter a number: "))
if num > 0 and set(str(num)) == {'1'}:
    print("Repunit Number")
else:
    print("Not a Repunit Number")
```

Determines if a number is a repunit number.

72. **Check for a Fascinating Number:**

```
num = int(input("Enter a number: "))
concatenated_num = int(str(num) + str(num*2) + str(num*3))
sorted_digits = sorted(str(concatenated_num))
if sorted_digits == ['1', '2', '3', '4', '5', '6', '7', '8', '9']:
    print("Fascinating Number")
else:
    print("Not a Fascinating Number")
```

Checks if a number is a fascinating number.

73. **Check for a Carmichael Number:**

```
num = int(input("Enter a number: "))
def is_carmichael(n):
    if n < 2 or n % 2 == 0:
        return False
    for a in range(2, n):
        if pow(a, n-1, n) != 1:
            return False
    return True
if is_carmichael(num):
    print("Carmichael Number")
else:
    print("Not a Carmichael Number")
```

Identifies if a number is a Carmichael number.

74. **Check for a Dihedral Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
rotations = [int(str(num)[i:] + str(num)[:i]) for i in range(len(str(num)))]
if all(is_prime(rotation) for rotation in rotations):
    print("Dihedral Prime Number")
else:
    print("Not a Dihedral Prime Number")
```

Checks if a number is a dihedral prime number.

75. **Check for a Lucky Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_lucky(n):
    sequence = [i for i in range(1, n + 1)]
    index = 1
    while sequence[index] <= n:
        sequence = sequence[:index+1] + sequence[index+2:]
        index += 1
        sequence = [sequence[i] for i in range(index, len(sequence))]
    return is_prime(sequence[-1])
if is_prime(num) and is_lucky(num):
    print("Lucky Prime Number")
else:
    print("Not a Lucky Prime Number")
```

Identifies if a number is a lucky prime number.

76. **Check for a Leyland Number:**

```
num = int(input("Enter a number: "))
def is_leyland(n, p, q):
    return n == pow(p, q) + pow(q, p)
for p in range(2, int(num**(1/2)) + 1):
    for q in range(2, int(num**(1/2)) + 1):
        if is_leyland(num, p, q):
            print("Leyland Number")
            break
else:
    print("Not a Leyland Number")
```

Checks if a number is a Leyland number.

77. **Check for a Centered Square Number:**

```
num = int(input("Enter a number: "))
centered_square_num = 1 + 4 * (num - 1) * (num - 2)
if centered_square_num == num * (2 * num - 1):
    print("Centered Square Number")
else:
    print("Not a Centered Square Number")
```

Determines if a number is a centered square number.

78. **Check for a Pancake Number:**

```
num = int(input("Enter a number: "))
def is_pancake(n):
    return bin(n)[2:].count('1') == 1
if is_pancake(num):
    print("Pancake Number")
else:
    print("Not a Pancake Number")
```

Identifies if a number is a pancake number.

79. **Check for a Permutable Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_permutable_prime(n):
    str_n = str(n)
    return all(is_prime(int(''.join(perm))) for perm in itertools.permutations(str_n))
if is_prime(num) and is_permutable_prime(num):
    print("Permutable Prime Number")
else:
    print("Not a Permutable Prime Number")
```

Checks if a number is a permutable prime number.

80. **Check for a Leyland Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_leyland_prime(n, p, q):
    return is_prime(pow(p, q) + pow(q, p))
for p in range(2, int(num**(1/2)) + 1):
    for q in range(2, int(num**(1/2)) + 1):
        if is_leyland_prime(num, p, q):
            print("Leyland Prime Number")
            break
else:
    print("Not a Leyland Prime Number")
```

Identifies if a number is a Leyland prime number.

81. **Check for a Tetrahedral Number:**

```
num = int(input("Enter a number: "))
tetrahedral_num = num * (num + 1) * (num + 2) // 6
if tetrahedral_num == num * (num + 1) * (2 * num + 1) // 6:
    print("Tetrahedral Number")
```

```
else:
    print("Not a Tetrahedral Number")
```

Checks if a number is a tetrahedral number.

82. **Check for a Fibonacci Tetrahedral Number:**

```
num = int
```

(input("Enter a number: ")) def is_fibonacci(n): a, b = 0, 1 while b < n: a, b = b, a + b return b == n tetrahedral_num = num * (num + 1) * (num + 2) // 6 if is_fibonacci(tetrahedral_num): print("Fibonacci Tetrahedral Number") else: print("Not a Fibonacci Tetrahedral Number") ``` Identifies if a number is a Fibonacci tetrahedral number.

83. **Check for a Sublime Number:**

```
num = int(input("Enter a number: "))
def is_sublime(n):
    return n == 2**n - 1
if is_sublime(num):
    print("Sublime Number")
else:
    print("Not a Sublime Number")
```

Determines if a number is a sublime number.

84. **Check for a Twisted Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_twisted_prime(n):
    str_n = str(n)
    return is_prime(int(str_n[::-1])) and is_prime(n)
if is_twisted_prime(num):
    print("Twisted Prime Number")
else:
    print("Not a Twisted Prime Number")
```

Checks if a number is a twisted prime number.

85. **Check for a Triangular Cube Number:**

```
num = int(input("Enter a number: "))
triangular_test = 8 * num + 1
cube_root = round(num**(1/3))
if math.isqrt(triangular_test)**2 == triangular_test and cube_root**3 == num:
    print("Triangular Cube Number")
else:
    print("Not a Triangular Cube Number")
```

Determines if a number is a triangular cube number.

86. **Check for a Dihedral Square Number:**

```
num = int(input("Enter a number: "))
def is_dihedral_square(n):
    str_n = str(n)
    return all(digit in ['0', '1', '8'] for digit in str_n)
if is_dihedral_square(num):
    print("Dihedral Square Number")
else:
    print("Not a Dihedral Square Number")
```

Checks if a number is a dihedral square number.

87. **Check for a Untouchable Number:**

```
num = int(input("Enter a number: "))
def sum_of_proper_divisors(n):
    return sum(i for i in range(1, n) if n % i == 0)
untouchable_set = set()
for i in range(2, num + 2):
    untouchable_set.add(sum_of_proper_divisors(i))
if num not in untouchable_set:
    print("Untouchable Number")
else:
    print("Not an Untouchable Number")
```

Identifies if a number is an untouchable number.

88. **Check for a Wolstenholme Prime Number:**

```
num = int(input("Enter a number: "))
def binomial_coefficient(n, k):
    if k > n:
        return 0
    result = 1
    for i in range(k):
        result *= (n - i)
        result //= (i + 1)
    return result
def is_wolstenholme_prime(n):
    return binomial_coefficient(2*n, n) % (n**3) == 0
if is_wolstenholme_prime(num):
    print("Wolstenholme Prime Number")
else:
    print("Not a Wolstenholme Prime Number")
```

Checks if a number is a Wolstenholme prime number.

89. **Check for a Schröder–Hipparchus Number:**

```
num = int(input("Enter a number: "))
def schroder_hipparchus_number(n):
    if n == 0:
        return 1
    return 3 * (2 * n - 1) * schroder_hipparchus_number(n - 1) // (n + 1)
if schroder_hipparchus_number(num) == math.factorial(num + 1):
    print("Schröder-Hipparchus Number")
else:
    print("Not a Schröder-Hipparchus Number")
```

Determines if a number is a Schröder–Hipparchus number.

90. **Check for a Superior Highly Composite Number:**

```
num = int(input("Enter a number: "))
def is_superior_highly_composite(n):
    count = 0
    for i in range(1, n + 1):
        if n % i == 0:
            count += 1
    return count > math.log2(math.log2(n))
if is_superior_highly_composite(num):
    print("Superior Highly Composite Number")
else:
    print("Not a Superior Highly Composite Number")
```

Checks if a number is a superior highly composite number.

Certainly! Here are 20 more Python if-else code snippets with short explanations:

91. **Check for a Power-ful Number:**

```
num = int(input("Enter a number: "))
def is_powerful_number(n):
    factors = set()
    for base in range(2, int(n**0.5) + 1):
        power = 2
        while base**power <= n:
            factors.add(base**power)
            power += 1
    return n in factors
if is_powerful_number(num):
    print("Power-ful Number")
else:
    print("Not a Power-ful Number")
```

Identifies if a number is a powerful number.

92. **Check for a Diophantine Equation Number:**

```
num = int(input("Enter a number: "))
def is_diophantine_equation_number(n):
    for x in range(1, n + 1):
        for y in range(1, n + 1):
            if x**2 + y**2 == n:
                return True
    return False
if is_diophantine_equation_number(num):
    print("Diophantine Equation Number")
else:
    print("Not a Diophantine Equation Number")
```

Checks if a number is a Diophantine equation number.

93. **Check for a Harshad Semiprime Number:**

```
num = int(input("Enter a number: "))
def is_harshad(n):
    return n % sum(int(digit) for digit in str(n)) == 0
def is_semiprime(n):
    count = 0
    for i in range(2, n + 1):
        while n % i == 0:
            count += 1
            n //= i
        if count > 2:
            return False
    return count == 2
if is_harshad(num) and is_semiprime(num):
    print("Harshad Semiprime Number")
```

```
else:
    print("Not a Harshad Semiprime Number")
```

Determines if a number is a Harshad semiprime number.

94. **Check for a Harshad Happy Number:**

```
num = int(input("Enter a number: "))
def is_harshad(n):
    return n % sum(int(digit) for digit in str(n)) == 0
def is_happy(n):
    seen = set()
    while n != 1 and n not in seen:
        seen.add(n)
        n = sum(int(digit)**2 for digit in str(n))
    return n == 1
if is_harshad(num) and is_happy(num):
    print("Harshad Happy Number")
else:
    print("Not a Harshad Happy Number")
```

Checks if a number is a Harshad happy number.

95. **Check for a Nearly Palindromic Number:**

```
num = int(input("Enter a number: "))
def is_palindrome(s):
    return s == s[::-1]
def is_nearly_palindromic(n):
    str_n = str(n)
    for i in range(len(str_n)):
        altered_n = str_n[:i] + str_n[i+1:]
        if is_palindrome(altered_n):
            return True
    return False
if is_nearly_palindromic(num):
    print("Nearly Palindromic Number")
else:
    print("Not a Nearly Palindromic Number")
```

Determines if a number is a nearly palindromic number.

96. **Check for a Partitioned Palindrome Number:**

```
num = int(input("Enter a number: "))
def is_palindrome(s):
    return s == s[::-1]
def is_partitioned_palindrome(n):
    str_n = str(n)
    for i in range(1, len(str_n)):
        left_partition = str_n[:i]
        right_partition = str_n[i:]
        if is_palindrome(left_partition) and is_palindrome(right_partition):
            return True
    return False
if is_partitioned_palindrome(num):
    print("Partitioned Palindrome Number")
else:
    print("Not a Partitioned Palindrome Number")
```

Checks if a number is a partitioned palindrome number.

97. **Check for a Fermat Prime Number:**

```
num = int(input("Enter a number: "))
def is_fermat_prime(n):
    return n == 2 or (2**(2**n) + 1) % n == 0
if is_fermat_prime(num):
    print("Fermat Prime Number")
else:
    print("Not a Fermat Prime Number")
```

Identifies if a number is a Fermat prime number.

98. **Check for a Circular Harshad Number:**

```
num = int(input("Enter a number: "))
def is_harshad(n):
    return n % sum(int(digit) for digit in str(n)) == 0
def is_circular_harshad(n):
    str_n = str(n)
    for i in range(len(str_n)):
        rotated_n = int(str_n[i:] + str_n[:i])
        if not is_harshad(rotated_n):
            return False
    return True
if is_circular_harshad(num):
    print("Circular Harshad Number")
else:
    print("Not a Circular Harshad Number")
```

Checks if a number is a circular Harshad number.

99. **Check for a Square Fibonacci Number:**

```python
num = int(input("Enter a number:"))
def is_fibonacci(n):
    a, b = 0, 1
    while b < n:
        a, b = b, a + b
    return b == n
if is_fibonacci(num) and math.isqrt(num)**2 == num:
    print("Square Fibonacci Number")
else:
    print("Not a Square Fibonacci Number")
```

Determines if a number is a square Fibonacci number.

100. **Check for a Generalized Fermat Prime Number:**

```python
num = int(input("Enter a number: "))
def is_generalized_fermat_prime(n):
    return n == 3 or (2**(2**n) + 1) % n == 0
if is_generalized_fermat_prime(num):
    print("Generalized Fermat Prime Number")
else:
    print("Not a Generalized Fermat Prime Number")
```

Identifies if a number is a generalized Fermat prime number.

101. **Check for a Hyperperfect Number:**

```python
num = int(input("Enter a number: "))
def is_hyperperfect(n):
    factors_sum = 1
    for i in range(2, n):
        if n % i == 0:
            factors_sum += i**(n // i)
    return factors_sum == 2 * n
if is_hyperperfect(num):
    print("Hyperperfect Number")
else:
    print("Not a Hyperperfect Number")
```

Checks if a number is a hyperperfect number.

102. **Check for a Solitary Prime Number:**

```python
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_solitary_prime(n):
    return is_prime(n) and not any(is_prime(int(str(n)[:i])) and is_prime(int(str(n)[i:]))
if is_solitary_prime(num):
    print("Solitary Prime Number")
else:
```

Identifies if a number is a solitary prime number.

103. *Check for a Bell Number:**

```
num = int(input("Enter a number: "))
bell_numbers = [1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975]
if num in bell_numbers:
    print("Bell Number")
else:
    print("Not a Bell Number")
```

Determines if a number is a Bell number.

104. *Check for a Wedderburn-Etherington Number:**

```
num = int(input("Enter a number: "))
def is_wedderburn_etherington(n):
    return n == 1 or 2**(n - 1) % n == 1
if is_wedderburn_etherington(num):
    print("Wedderburn-Etherington Number")
else:
    print("Not a Wedderburn-Etherington Number")
```

Identifies if a number is a Wedderburn-Etherington number.

105. *Check for a Juggler Sequence Number:**

```
num = int(input("Enter a number: "))
def is_juggler_sequence(n):
    while n > 1:
        if n % 2 == 0:
            n = math.isqrt(n)
        else:
            n = math.isqrt(n**3)
    return n == 1
if is_juggler_sequence(num):
    print("Juggler Sequence Number")
```

```
else:
    print("Not a Juggler Sequence Number")
```

Checks if a number is a Juggler sequence number.

106. *Check for a Balanced Prime Number:**

```
num = int(input("Enter a number: "))
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_balanced_prime(n):
    str_n = str(n)
    return all(is_prime(int(digit)) for digit in str_n)
if is_balanced_prime(num):
    print("Balanced Prime Number")
else:
    print("Not a Balanced Prime Number")
```

Identifies if a number is a balanced prime number.