For Loop:

1. **Print numbers from 1 to 10 using a for loop:**
   - for i in range(1, 11):
   - print(i)
2. **Difference between a for loop and a while loop in Python:**
   - **For Loop:** Iterates over a sequence (like a list, tuple, dictionary, set, or string) or a range of numbers.
   - **While Loop:** Repeats as long as a certn condition is true.
3. **Calculate the sum of all numbers from 1 to 100 using a for loop:**
   - total = 0
   - for i in range(1, 101):
   - total += i
   - print(total)
4. **Iterate through a list using a for loop:**

```
my_list = [1, 2, 3, 4, 5]
for item in my_list:
  print(item)
```

5. **Find the product of all elements in a list using a for loop:**

```
my_list = [1, 2, 3, 4, 5]
product = 1
for item in my_list:
  product *= item
print(product)
```

6. **Print all even numbers from 1 to 20 using a for loop:**

```
for i in range(1, 21):
  if i % 2 == 0:
    print(i)
```

7. **Calculate the factorial of a number using a for loop:**

```
num = 5
factorial = 1
for i in range(1, num + 1):
  factorial *= i
print(factorial)
```

8. **Iterate through the characters of a string using a for loop:**

```
my_string = "hello"
for char in my_string:
  print(char)
```

9. **Find the largest number in a list using a for loop:**

```
my_list = [1, 2, 3, 4, 5]
largest = my_list[0]
for item in my_list:
  if item > largest:
    largest = item
print(largest)
```

10. **Print the Fibonacci sequence up to a specified limit using a for loop:**

```
limit = 10
a, b = 0, 1
for _ in range(limit):
  print(a)
  a, b = b, a + b
```

11. **Count the number of vowels in a given string using a for loop:**

```
my_string = "hello world"
vowels = "aeiou"
count = 0
```

```
for char in my_string:
    if char in vowels:
        count += 1
print(count)
```

12. **Generate a multiplication table for a given number using a for loop:**

```
num = 5
for i in range(1, 11):
    print(f"{num} x {i} = {num * i}")
```

13. **Reverse a list using a for loop:**

```
my_list = [1, 2, 3, 4, 5]
reversed_list = []
for item in my_list:
    reversed_list.insert(0, item)
print(reversed_list)
```

14. **Find the common elements between two lists using a for loop:**

```
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
common_elements = []
for item in list1:
    if item in list2:
        common_elements.append(item)
print(common_elements)
```

15. **Iterate through the keys and values of a dictionary using a for loop:**

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
for key, value in my_dict.items():
    print(f"Key: {key}, Value: {value}")
```

16. **Find the GCD (Greatest Common Divisor) of two numbers using a for loop:**

```
a, b = 60, 48
while b:
    a, b = b, a % b
print(a)
```

17. **Check if a string is a palindrome using a for loop:**

```
my_string = "radar"
is_palindrome = True
for i in range(len(my_string) // 2):
    if my_string[i] != my_string[-(i + 1)]:
        is_palindrome = False
        break
print(is_palindrome)
```

18. **Remove duplicates from a list using a for loop:**

```
my_list = [1, 2, 2, 3, 4, 4, 5]
unique_list = []
for item in my_list:
    if item not in unique_list:
        unique_list.append(item)
print(unique_list)
```

19. **Count the number of words in a sentence using a for loop:**

```
sentence = "This is a sample sentence"
words = sentence.split()
word_count = 0
for word in words:
    word_count += 1
print(word_count)
```

20. **Find the sum of all odd numbers from 1 to 50 using a for loop:**

```
total = 0
```

```
for i in range(1, 51):
    if i % 2 != 0:
        total += i
print(total)
```

21. Check if a given year is a leap year using a for loop:

```
year = 2024
is_leap = False
if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    is_leap = True
print(is_leap)
```

22. Calculate the square root of a number using a for loop:

```
num = 16
for i in range(num):
    if i * i == num:
        print(i)
        break
```

23. Find the LCM (Least Common Multiple) of two numbers using a for loop:

```
a, b = 12, 15
greater = max(a, b)
while True:
    if greater % a == 0 and greater % b == 0:
        lcm = greater
        break
    greater += 1
print(lcm)
```

If-else:

Check if a number is positive, negative, or zero:

```
num = float(input("Enter a number: "))
if num > 0:
    print("The number is positive.")
elif num < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

Check if a number is even or odd:

```
num = int(input("Enter a number: "))
if num % 2 == 0:
    print("The number is even.")
else:
    print("The number is odd.")
```

Nested if-else statements example:

```
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("The number is zero.")
    else:
        print("The number is positive.")
else:
    print("The number is negative.")
```

Determine the largest of three numbers:

```
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))
if num1 >= num2 and num1 >= num3:
    largest = num1
```

```
elif num2 >= num1 and num2 >= num3:
   largest = num2
else:
   largest = num3
print("The largest number is", largest)
```

Calculate the absolute value of a number:

```
num = float(input("Enter a number: "))
if num >= 0:
   abs_value = num
else:
   abs_value = -num
print("The absolute value is", abs_value)
```

Check if a character is a vowel or consonant:

```
char = input("Enter a character: ").lower()
if char in 'aeiou':
   print("The character is a vowel.")
else:
   print("The character is a consonant.")
```

Determine if a user is eligible to vote based on age:

```
age = int(input("Enter your age: "))
if age >= 18:
   print("You are eligible to vote.")
else:
   print("You are not eligible to vote.")
```

Calculate the discount amount based on the purchase amount:

```
purchase_amount = float(input("Enter the purchase amount: "))
if purchase_amount >= 1000:
   discount = purchase_amount * 0.1
else:
   discount = purchase_amount * 0.05
print("The discount amount is", discount)
```

Check if a number is within a specified range:

```
num = float(input("Enter a number: "))
lower_bound = float(input("Enter the lower bound: "))
upper_bound = float(input("Enter the upper bound: "))
if lower_bound <= num <= upper_bound:
   print("The number is within the range.")
else:
   print("The number is outside the range.")
```

Determine the grade of a student based on their score:

```
score = float(input("Enter the score: "))
if score >= 90:
   grade = 'A'
elif score >= 80:
   grade = 'B'
elif score >= 70:
   grade = 'C'
elif score >= 60:
   grade = 'D'
else:
   grade = 'F'
print("The grade is", grade)
```

Check if a string is empty or not:

```
string = input("Enter a string: ")
if string:
```

```
    print("The string is not empty.")
else:
    print("The string is empty.")
```

Identify the type of a triangle:

```
side1 = float(input("Enter the first side: "))
side2 = float(input("Enter the second side: "))
side3 = float(input("Enter the third side: "))
if side1 == side2 == side3:
    print("The triangle is equilateral.")
elif side1 == side2 or side2 == side3 or side1 == side3:
    print("The triangle is isosceles.")
else:
    print("The triangle is scalene.")
```

Determine the day of the week based on a number:

```
day_num = int(input("Enter a number (1-7): "))
if day_num == 1:
    day = "Monday"
elif day_num == 2:
    day = "Tuesday"
elif day_num == 3:
    day = "Wednesday"
elif day_num == 4:
    day = "Thursday"
elif day_num == 5:
    day = "Friday"
elif day_num == 6:
    day = "Saturday"
elif day_num == 7:
    day = "Sunday"
else:
    day = "Invalid number"
print("The day is", day)
```

Check if a year is a leap year using if-else and a function:

```
def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

year = int(input("Enter a year: "))
if is_leap_year(year):
    print(year, "is a leap year.")
else:
    print(year, "is not a leap year.")
```

Use the “assert” statement to add debugging checks:

```
num = int(input("Enter a number: "))
assert num >= 0, "Number must be non-negative"
if num % 2 == 0:
    print("The number is even.")
else:
    print("The number is odd.")
```

Determine eligibility for a senior citizen discount based on age:

```
age = int(input("Enter your age: "))
if age >= 60:
    print("You are eligible for a senior citizen discount.")
else:
```

```
    print("You are not eligible for a senior citizen discount.")
```

Categorize a given character as uppercase, lowercase, or neither:

```
char = input("Enter a character: ")
if char.isupper():
    print("The character is uppercase.")
elif char.islower():
    print("The character is lowercase.")
else:
    print("The character is neither uppercase nor lowercase.")
```

Determine the roots of a quadratic equation:

```
import math

a = float(input("Enter coefficient a: "))
b = float(input("Enter coefficient b: "))
c = float(input("Enter coefficient c: "))

discriminant = b**2 - 4*a*c

if discriminant > 0:
    root1 = (-b + math.sqrt(discriminant)) / (2*a)
    root2 = (-b - math.sqrt(discriminant)) / (2*a)
    print("The roots are real and different:", root1, root2)
elif discriminant == 0:
    root = -b / (2*a)
    print("The roots are real and the same:", root)
else:
    real_part = -b / (2*a)
    imaginary_part = math.sqrt(-discriminant) / (2*a)
    print("The roots are complex and different:", real_part, "+", imaginary_part, "i and", real_part, "-",
imaginary_part, "i")
```

Check if a given year is a century year:

```
year = int(input("Enter a year: "))
if year % 100 == 0:
    print(year, "is a century year.")
else:
    print(year, "is not a century year.")
```

Determine if a given number is a perfect square:

```
import math

num = int(input("Enter a number: "))
sqrt = math.isqrt(num)
if sqrt * sqrt == num:
    print(num, "is a perfect square.")
else:
    print(num, "is not a perfect square.")
```

Purpose of the “continue” and “break” statements within if-else loops:

continue: Skips the rest of the code inside the loop for the current iteration and moves to the next iteration.

break: Exits the loop immediately, skipping all remaining iterations.

Example:

```
for i in range(1, 11):
        if i == 5:
        continue  # Skip the rest of the code for this iteration
        if i == 8:
        break  # Exit the loop
```

```
            print(i)
```

Calculate the BMI (Body Mass Index) of a person:

```
weight = float(input("Enter your weight in kilograms: "))
height = float(input("Enter your height in meters: "))
bmi = weight / (height ** 2)

if bmi < 18.5:
   category = "Underweight"
elif 18.5 <= bmi < 24.9:
   category = "Normal weight"
elif 25 <= bmi < 29.9:
   category = "Overweight"
else:
   category = "Obesity"

print("Your BMI is", bmi, "and you are categorized as", category)
```

Use the filter() function with if-else statements to filter elements from a list:

```
def is_even(num):
   return num % 2 == 0

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = list(filter(is_even, numbers))
print("Even numbers:", even_numbers)
```

Determine if a given number is prime:

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

Map:

Purpose of the map() function: The map() function applies a given function to each item of an iterable (like a list) and returns a map object (an iterator). It is useful for transforming data without using explicit loops.

Example:

```
def square(x):
        return x * x

numbers = [1, 2, 3, 4, 5]
squared_numbers = map(square, numbers)
print(list(squared_numbers))  # Output: [1, 4, 9, 16, 25]
```

Square each element of a list of numbers:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(lambda x: x ** 2, numbers)
print(list(squared_numbers))  # Output: [1, 4, 9, 16, 25]
```

Difference between map() and list comprehension:

map() applies a function to all items in an input list and returns an iterator.

List comprehension is more flexible and can include conditionals.

Example using list comprehension:

```
numbers = [1, 2, 3, 4, 5]
```

```
squared_numbers = [x ** 2 for x in numbers]
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]
```

When to choose one over the other:

Use map() when you have a function ready to apply to all elements.

Use list comprehension for more complex transformations or when you need conditionals.

Convert a list of names to uppercase:

```
names = ["alice", "bob", "charlie"]
uppercase_names = map(str.upper, names)
print(list(uppercase_names))  # Output: ['ALICE', 'BOB', 'CHARLIE']
```

Calculate the length of each word in a list of strings:

```
words = ["hello", "world", "python"]
lengths = map(len, words)
print(list(lengths))  # Output: [5, 5, 6]
```

Apply a custom function to elements of multiple lists simultaneously:

```
def add(x, y):
   return x + y

list1 = [1, 2, 3]
list2 = [4, 5, 6]
summed_list = map(add, list1, list2)
print(list(summed_list))  # Output: [5, 7, 9]
```

Convert a list of temperatures from Celsius to Fahrenheit:

```
def celsius_to_fahrenheit(c):
   return (c * 9/5) + 32

celsius_temps = [0, 20, 37, 100]
fahrenheit_temps = map(celsius_to_fahrenheit, celsius_temps)
print(list(fahrenheit_temps))  # Output: [32.0, 68.0, 98.6, 212.0]
```

Round each element of a list of floating-point numbers to the nearest integer:

```
float_numbers = [1.2, 2.5, 3.7, 4.4]
rounded_numbers = map(round, float_numbers)
print(list(rounded_numbers))  # Output: [1, 2, 4, 4]
```

Reduce:

What is the reduce() function in Python, and what module should you import to use it?

Provide an example of its basic usage: The reduce() function is used to apply a function cumulatively to the items of an iterable, from left to right, so as to reduce the iterable to a single value. It is part of the functools module.

Example:

```
from functools import reduce

def add(x, y):
        return x + y

numbers = [1, 2, 3, 4, 5]
result = reduce(add, numbers)
print(result)  # Output: 15
```

Find the product of all elements in a list:

```
from functools import reduce

numbers = [1, 2, 3, 4, 5]
product = reduce(lambda x, y: x * y, numbers)
print(product)  # Output: 120
```

Find the maximum element in a list of numbers:

```
from functools import reduce

numbers = [1, 5, 3, 9, 2]
maximum = reduce(lambda x, y: x if x > y else y, numbers)
print(maximum)  # Output: 9
```

Concatenate a list of strings into a single string:

```
from functools import reduce

strings = ["Hello", " ", "world", "!"]
concatenated = reduce(lambda x, y: x + y, strings)
print(concatenated)  # Output: "Hello world!"
```

Calculate the factorial of a number:

```
from functools import reduce

def factorial(n):
   return reduce(lambda x, y: x * y, range(1, n + 1))

number = 5
print(factorial(number))  # Output: 120
```

Find the GCD (Greatest Common Divisor) of a list of numbers:

```
from functools import reduce
import math

numbers = [48, 64, 16]
gcd = reduce(math.gcd, numbers)
print(gcd)  # Output: 16
```

Find the sum of the digits of a given number:

```
from functools import reduce

number = 12345
sum_of_digits = reduce(lambda x, y: x + y, map(int, str(number)))
print(sum_of_digits)  # Output: 15
```

Filter

Purpose of the filter() function and an example: The filter() function constructs an iterator from elements of an iterable for which a function returns true.

Example:

```
def is_even(x):
        return x % 2 == 0

numbers = [1, 2, 3, 4, 5, 6]
even_numbers = filter(is_even, numbers)
print(list(even_numbers))  # Output: [2, 4, 6]
```

Select even numbers from a list of integers:

```
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = filter(lambda x: x % 2 == 0, numbers)
print(list(even_numbers))  # Output: [2, 4, 6]
```

Select names that start with a specific letter:

```
names = ["Alice", "Bob", "Charlie", "David"]
specific_letter = 'A'
filtered_names = filter(lambda name: name.startswith(specific_letter), names)
print(list(filtered_names))  # Output: ['Alice']
```

Select prime numbers from a list of integers:

```
def is_prime(n):
```

```
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

numbers = [2, 3, 4, 5, 6, 7, 8, 9, 10]
prime_numbers = filter(is_prime, numbers)
print(list(prime_numbers))  # Output: [2, 3, 5, 7]
```

Remove None values from a list:

```
values = [1, None, 2, None, 3, 4, None]
non_none_values = filter(None, values)
print(list(non_none_values))  # Output: [1, 2, 3, 4]
```

Select words longer than a certain length:

```
words = ["hello", "world", "python", "is", "awesome"]
min_length = 5
long_words = filter(lambda word: len(word) > min_length, words)
print(list(long_words))  # Output: ['python', 'awesome']
```

Select elements greater than a specified threshold:

```
values = [10, 20, 30, 40, 50]
threshold = 25
filtered_values = filter(lambda x: x > threshold, values)
print(list(filtered_values))  # Output: [30, 40, 50]
```

Recursion:

Concept of Recursion in Python: Recursion is a programming technique where a function calls itself in order to solve a problem. It differs from iteration in that recursion involves function calls and a base case to terminate the recursive calls, whereas iteration uses loops to repeat a block of code.

Calculate the factorial of a number using recursion:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

number = 5
print(factorial(number))  # Output: 120
```

Find the nth Fibonacci number using recursion:

```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

n = 10
print(fibonacci(n))  # Output: 55
```

Calculate the sum of all elements in a list using recursion:

```
def sum_list(lst):
    if not lst:
        return 0
    else:
```

```
    return lst[0] + sum_list(lst[1:])

numbers = [1, 2, 3, 4, 5]
print(sum_list(numbers))  # Output: 15
```

Prevent a recursive function from running indefinitely: To prevent a stack overflow error, ensure that your recursive function has a base case that will eventually be met, terminating the recursion.

Find the greatest common divisor (GCD) using the Euclidean algorithm:

```
def gcd(a, b):
  if b == 0:
    return a
  else:
    return gcd(b, a % b)

print(gcd(48, 18))  # Output: 6
```

Reverse a string using recursion:

```
def reverse_string(s):
  if len(s) == 0:
    return s
  else:
    return s[-1] + reverse_string(s[:-1])

print(reverse_string("hello"))  # Output: "olleh"
```

Calculate the power of a number (x^n) using recursion:

```
def power(x, n):
  if n == 0:
    return 1
  else:
    return x * power(x, n - 1)

print(power(2, 3))  # Output: 8
```

Find all permutations of a given string using recursion:

```
def permutations(s):
  if len(s) == 1:
    return [s]
  else:
    perm_list = []
    for i in range(len(s)):
      part = s[:i] + s[i+1:]
      for p in permutations(part):
        perm_list.append(s[i] + p)
    return perm_list

print(permutations("abc"))  # Output: ['abc', 'acb', 'bac', 'bca', 'cab', 'cba']
```

Check if a string is a palindrome using recursion:

```
def is_palindrome(s):
  if len(s) <= 1:
    return True
  elif s[0] != s[-1]:
    return False
  else:
    return is_palindrome(s[1:-1])

print(is_palindrome("radar"))  # Output: True
```

Generate all possible combinations of a list of elements using recursion:

```
def combinations(lst):
    if len(lst) == 0:
        return [[]]
    else:
        combs = []
        for c in combinations(lst[1:]):
            combs += [c, [lst[0]] + c]
        return combs

print(combinations([1, 2, 3]))  # Output: [[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]
```

Basics of Functions

What is a function in Python, and why is it used? A function is a block of organized, reusable code that performs a single action. Functions help to break down complex problems into smaller, manageable pieces, promote code reuse, and improve readability.

How do you define a function in Python? Provide an example:

```
def greet(name):
    return f"Hello, {name}!"

print(greet("Alice"))  # Output: "Hello, Alice!"
```

Difference between a function definition and a function call:

Function Definition: The code that specifies what the function does.

Function Call: The code that executes the function.

Example:

```
def add(a, b):  # Function definition
        return a + b

result = add(2, 3)  # Function call
print(result)  # Output: 5
```

Calculate the sum of two numbers and call the function:

```
def sum_numbers(a, b):
    return a + b

print(sum_numbers(3, 4))  # Output: 7
```

What is a function signature, and what information does it typically include? A function signature includes the function name, the number and types of parameters, and the return type. It provides a summary of how to call the function and what to expect.

Create a function that takes two arguments and returns their product:

```
def multiply(a, b):
    return a * b

print(multiply(3, 4))  # Output: 12
```

Function Parameters and Arguments

Formal Parameters and Actual Arguments:

Formal Parameters: These are the variables defined in the function declaration/definition. They act as placeholders for the values that will be passed to the function.

Actual Arguments: These are the real values or variables passed to the function when it is called.

Example:

```
def greet(name):  # 'name' is a formal parameter
```

```
        print(f"Hello, {name}!")

    greet("Alice")  # "Alice" is an actual argument
```

Function with Default Argument Values:

```
def greet(name="Guest"):
  print(f"Hello, {name}!")

greet()  # Output: Hello, Guest!
greet("Alice")  # Output: Hello, Alice!
```

Keyword Arguments: Keyword arguments allow you to specify the value of a parameter by name, making the function call more readable.

Example:

```
def greet(name, message):
    print(f"{message}, {name}!")

greet(name="Alice", message="Good morning")  # Output: Good morning, Alice!
greet(message="Hello", name="Bob")  # Output: Hello, Bob!
```

Function that Accepts a Variable Number of Arguments:

```
def sum_numbers(*args):
  return sum(args)

print(sum_numbers(1, 2, 3))  # Output: 6
print(sum_numbers(4, 5, 6, 7, 8))  # Output: 30
```

Purpose of *args and **kwargs:

*args allows a function to accept any number of positional arguments.

**kwargs allows a function to accept any number of keyword arguments.

Example:

```
def example_function(*args, **kwargs):
    print("Positional arguments:", args)
    print("Keyword arguments:", kwargs)

example_function(1, 2, 3, a=4, b=5)
# Output:
# Positional arguments: (1, 2, 3)
# Keyword arguments: {'a': 4, 'b': 5}
```

Return Values and Scoping

Role of the return Statement: The return statement is used to exit a function and return a value to the caller. It can return any type of value, including None.

Examples:

```
def add(a, b):
    return a + b

result = add(3, 4)
print(result)  # Output: 7

def no_return():
    print("This function returns None by default")

print(no_return())  # Output: This function returns None by default
                    #         None
```

Variable Scope:

Local Variables: Variables defined within a function and accessible only within that function.

Global Variables: Variables defined outside any function and accessible throughout the program.

Example:

```
x = 10  # Global variable

def example_function():
    y = 5  # Local variable
    print("Inside function, x:", x)
    print("Inside function, y:", y)

example_function()
print("Outside function, x:", x)
# print("Outside function, y:", y)  # This would cause an error because y is not defined outside the function
```

Use of Global Variables within Functions:

```
x = 10

def modify_global():
  global x
  x = 20

print("Before function call, x:", x)  # Output: 10
modify_global()
print("After function call, x:", x)  # Output: 20
```

Calculate the Factorial of a Number and Return It:

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```

Access Variables Defined Outside a Function: You can access global variables directly within a function, but to modify them, you need to use the global keyword.

Example:

```
x = 10

def access_global():
    print("Accessing global variable x:", x)

access_global()  # Output: Accessing global variable x: 10
```

Lambda Functions and Higher-Order Functions

Lambda Functions: Lambda functions are small anonymous functions defined using the lambda keyword. They are typically used for short, simple operations and are often used as arguments to higher-order functions like map(), filter(), and reduce().

Example:

```
add = lambda x, y: x + y
print(add(2, 3))  # Output: 5
```

Sort a List of Tuples Based on the Second Element:

```
tuples = [(1, 3), (4, 1), (5, 2)]
sorted_tuples = sorted(tuples, key=lambda x: x[1])
print(sorted_tuples)  # Output: [(4, 1), (5, 2), (1, 3)]
```

Higher-Order Functions: Higher-order functions are functions that take other functions as arguments or return functions as their result. They are useful for creating more abstract and reusable code.

Example:

```
def apply_function(func, value):
    return func(value)

def square(x):
    return x * x

print(apply_function(square, 5))  # Output: 25
```

Function that Takes a List of Numbers and a Function as Arguments:

```
def apply_to_each(numbers, func):
  return [func(num) for num in numbers]

numbers = [1, 2, 3, 4, 5]
squared_numbers = apply_to_each(numbers, lambda x: x ** 2)
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]
```

Built-in Functions

Role of Built-in Functions:

len(): Returns the length of an object.
max(): Returns the largest item in an iterable or the largest of two or more arguments.
min(): Returns the smallest item in an iterable or the smallest of two or more arguments.

Examples:

```
print(len("hello"))  # Output: 5
print(max([1, 2, 3, 4, 5]))  # Output: 5
print(min(3, 1, 4, 2))  # Output: 1
```

Use map() to Apply a Function to Each Element of a List:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(lambda x: x ** 2, numbers)
print(list(squared_numbers))  # Output: [1, 4, 9, 16, 25]
```

How filter() Works and When to Use It: The filter() function constructs an iterator from elements of an iterable for which a function returns true. It is used to filter out elements based on a condition.

Example:

```
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = filter(lambda x: x % 2 == 0, numbers)
print(list(even_numbers))  # Output: [2, 4, 6]
```

Use reduce() to Find the Product of All Elements in a List:

```
from functools import reduce

numbers = [1, 2, 3, 4, 5]
product = reduce(lambda x, y: x * y, numbers)
print(product)  # Output: 120
```

Function Documentation and Best Practices

Purpose of Docstrings: Docstrings are used to document what a function does. They are written as the first statement in a function and can be accessed using the __doc__ attribute.

Example:

```
def add(a, b):
    """
    Add two numbers and return the result.
```

```
        Parameters:
        a (int): The first number.
        b (int): The second number.

        Returns:
        int: The sum of the two numbers.
        """
        return a + b

print(add.__doc__)
```

Best Practices for Naming Functions and Variables:

Use descriptive names that convey the purpose of the function or variable.
Follow the PEP 8 naming conventions: use lowercase words separated by underscores for functions and variables.
Avoid using single-character names except for loop counters or indices.
Use consistent naming patterns throughout your code.

Example:

```
def calculate_area(radius):
        pi = 3.14159
        return pi * radius ** 2

area = calculate_area(5)
print(area)  # Output: 78.53975
```