SMART INTERNZ-APSCHE

Date: February 14, 2024

Al/ML. Training

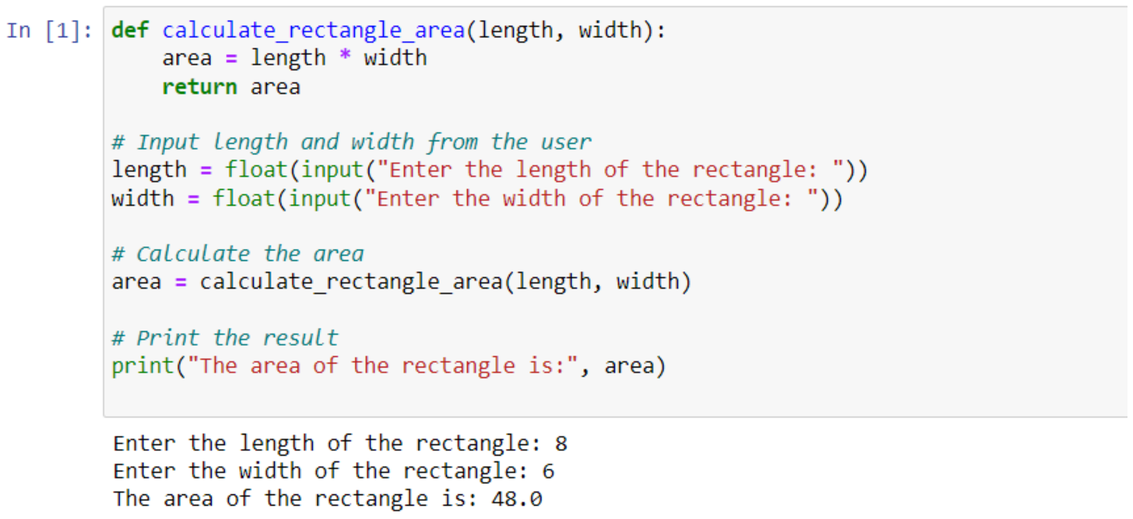
Assessment-1

1. Write a Python program to calculate the area of a rectangle given its length and width.

Program:

```
def calculate_rectangle_area(length, width):
   area = length * width
   return area
# Input length and width from the user
length = float(input("Enter the length of the rectangle: "))
width = float(input("Enter the width of the rectangle: "))
# Calculate the area
area = calculate_rectangle_area(length, width)
# Print the result
print("The area of the rectangle is:", area)
```

Output:



```
In [1]: def calculate_rectangle_area(length, width):
            area = length * width
            return area

        # Input length and width from the user
        length = float(input("Enter the length of the rectangle: "))
        width = float(input("Enter the width of the rectangle: "))

        # Calculate the area
        area = calculate_rectangle_area(length, width)

        # Print the result
        print("The area of the rectangle is:", area)
```

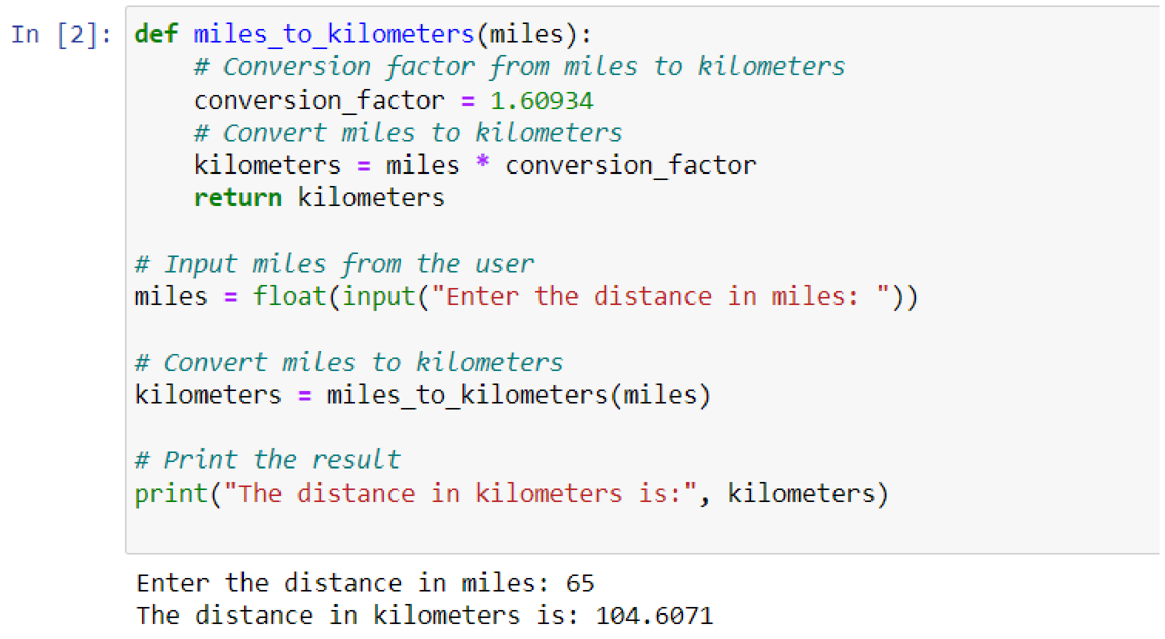
```
Enter the length of the rectangle: 8
Enter the width of the rectangle: 6
The area of the rectangle is: 48.0
```

2. Write a program to convert miles to kilometers.

Program:

```
def miles_to_kilometers(miles):
    # Conversion factor from miles to kilometers
    conversion_factor = 1.60934
    # Convert miles to kilometers
    kilometers = miles * conversion_factor
    return kilometers
# Input miles from the user
miles = float(input("Enter the distance in miles: "))
# Convert miles to kilometers
kilometers = miles_to_kilometers(miles)
# Print the result
print("The distance in kilometers is:", kilometers)
```

Output:



```
In [2]: def miles_to_kilometers(miles):
            # Conversion factor from miles to kilometers
            conversion_factor = 1.60934
            # Convert miles to kilometers
            kilometers = miles * conversion_factor
            return kilometers

        # Input miles from the user
        miles = float(input("Enter the distance in miles: "))

        # Convert miles to kilometers
        kilometers = miles_to_kilometers(miles)

        # Print the result
        print("The distance in kilometers is:", kilometers)
```

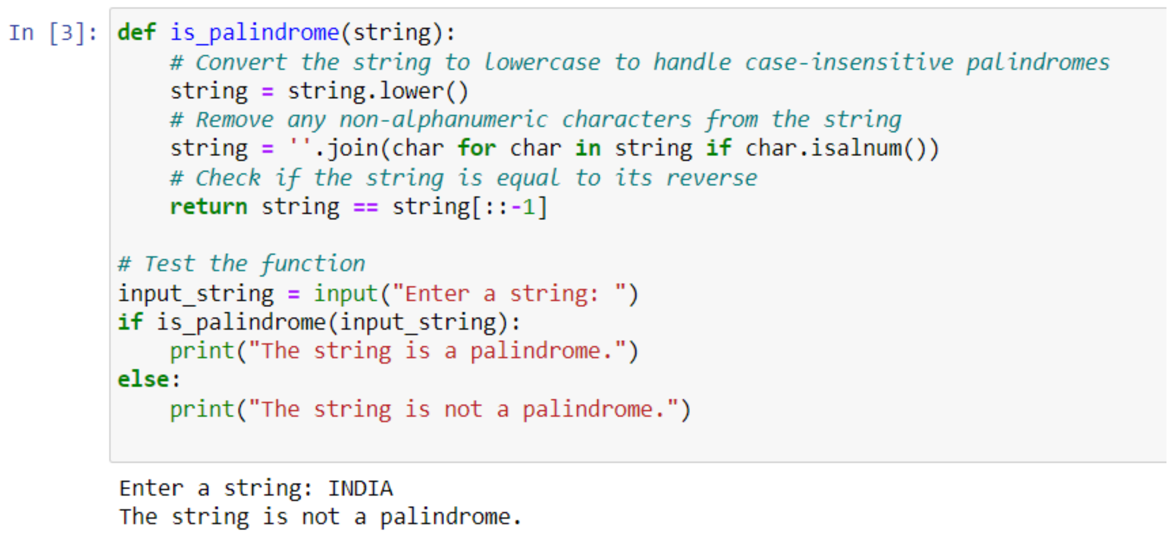
```
Enter the distance in miles: 65
The distance in kilometers is: 104.6071
```

3. Write a function to check if a given string is a palindrome.

Program:

```
def is_palindrome(string):
    # Convert the string to lowercase to handle case-insensitive palindromes
    string = string.lower()
    # Remove any non-alphanumeric characters from the string
    string = ''.join(char for char in string if char.isalnum())
    # Check if the string is equal to its reverse
    return string == string[::-1]
# Test the function
input_string = input("Enter a string: ")
if is_palindrome(input_string):
    print("The string is a palindrome.")
else:
    print("The string is not a palindrome.")
```

Output:



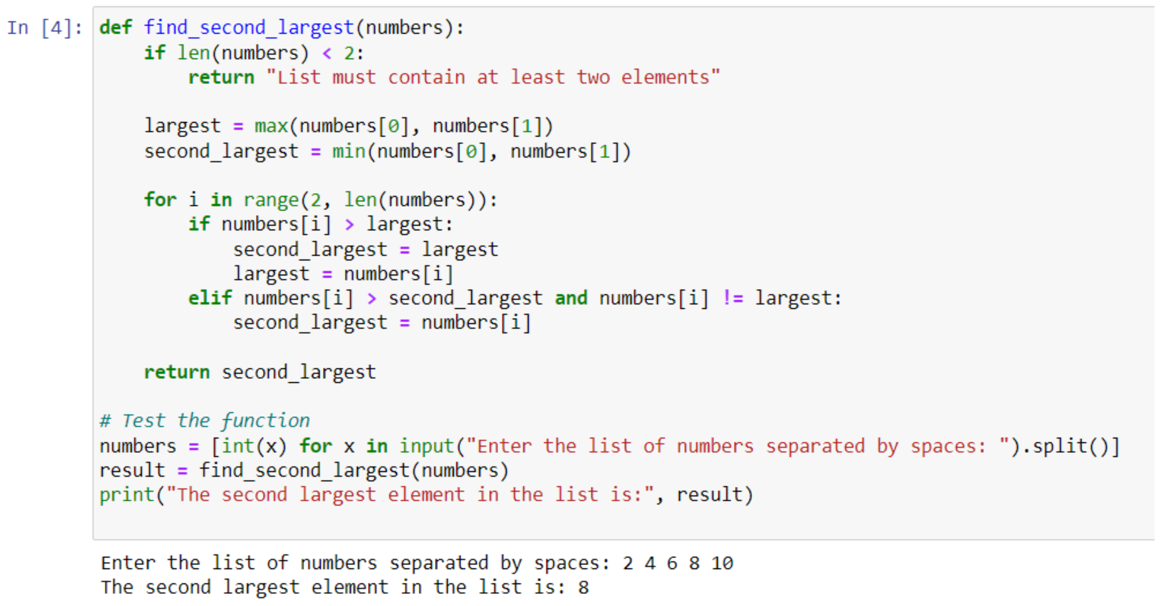
```
In [3]: def is_palindrome(string):
            # Convert the string to lowercase to handle case-insensitive palindromes
            string = string.lower()
            # Remove any non-alphanumeric characters from the string
            string = ''.join(char for char in string if char.isalnum())
            # Check if the string is equal to its reverse
            return string == string[::-1]

        # Test the function
        input_string = input("Enter a string: ")
        if is_palindrome(input_string):
            print("The string is a palindrome.")
        else:
            print("The string is not a palindrome.")

        Enter a string: INDIA
        The string is not a palindrome.
```

4. Write a Python program to find the second largest element in a list.

Program:

```
def find_second_largest(numbers):
   if len(numbers) < 2:
      return "List must contain at least two elements"
   largest = max(numbers[0], numbers[1])
   second_largest = min(numbers[0], numbers[1])
   for i in range(2, len(numbers)):
      if numbers[i] > largest:
         second_largest = largest
         largest = numbers[i]
      elif numbers[i] > second_largest and numbers[i] != largest:
         second_largest = numbers[i]
   return second_largest
# Test the function
numbers = [int(x) for x in input("Enter the list of numbers separated by spaces:
").split()]
result = find_second_largest(numbers)
print("The second largest element in the list is:", result)
```

Output:



```
In [4]: def find_second_largest(numbers):
            if len(numbers) < 2:
                return "List must contain at least two elements"

            largest = max(numbers[0], numbers[1])
            second_largest = min(numbers[0], numbers[1])

            for i in range(2, len(numbers)):
                if numbers[i] > largest:
                    second_largest = largest
                    largest = numbers[i]
                elif numbers[i] > second_largest and numbers[i] != largest:
                    second_largest = numbers[i]

            return second_largest

        # Test the function
        numbers = [int(x) for x in input("Enter the list of numbers separated by spaces: ").split()]
        result = find_second_largest(numbers)
        print("The second largest element in the list is:", result)

        Enter the list of numbers separated by spaces: 2 4 6 8 10
        The second largest element in the list is: 8
```

5. Explain what indentation means in Python.

In Python, indentation refers to the use of whitespace (typically spaces or tabs) at the beginning of a line to define the structure and scope of code. Unlike many other programming languages that use braces or keywords to denote blocks of code (such as {} in C, C++, Java, etc.), Python uses indentation to indicate the beginning and end of blocks.

Indentation is crucial in Python because it determines the grouping of statements. It helps Python to understand the hierarchy and flow of control in the code. Here are some key points about indentation in Python:

Block Structure: Indentation is used to define the block structure of code, such as loops, conditional statements, function definitions, class definitions, etc. Blocks of code at the same level of indentation are considered to be part of the same block.

Consistency: Python relies on consistent indentation to interpret the code correctly. Mixing spaces and tabs for indentation can lead to errors or unexpected behavior.

Indentation Level: The standard convention in Python is to use four spaces for each level of indentation. This is recommended by the official Python style guide (PEP 8). While you can technically use any consistent indentation level, four spaces is the most widely adopted convention.

Whitespace Significance: In Python, whitespace (including indentation) is significant and is used to determine the beginning and end of statements, as well as the structure of blocks of code. Incorrect indentation can lead to syntax errors or changes in the logical meaning of the code.

No Mandatory Braces: Unlike languages like C, Java, or JavaScript, Python does not use braces {} to define blocks of code. Instead, indentation serves this purpose. This results in more readable and visually consistent code, but it also requires careful attention to indentation levels.

Here's an example of indentation in Python code:

```
if x > 0:
    print("x is positive")
    if y > 0:
        print("y is also positive")
else:
    print("x is non-positive")
```

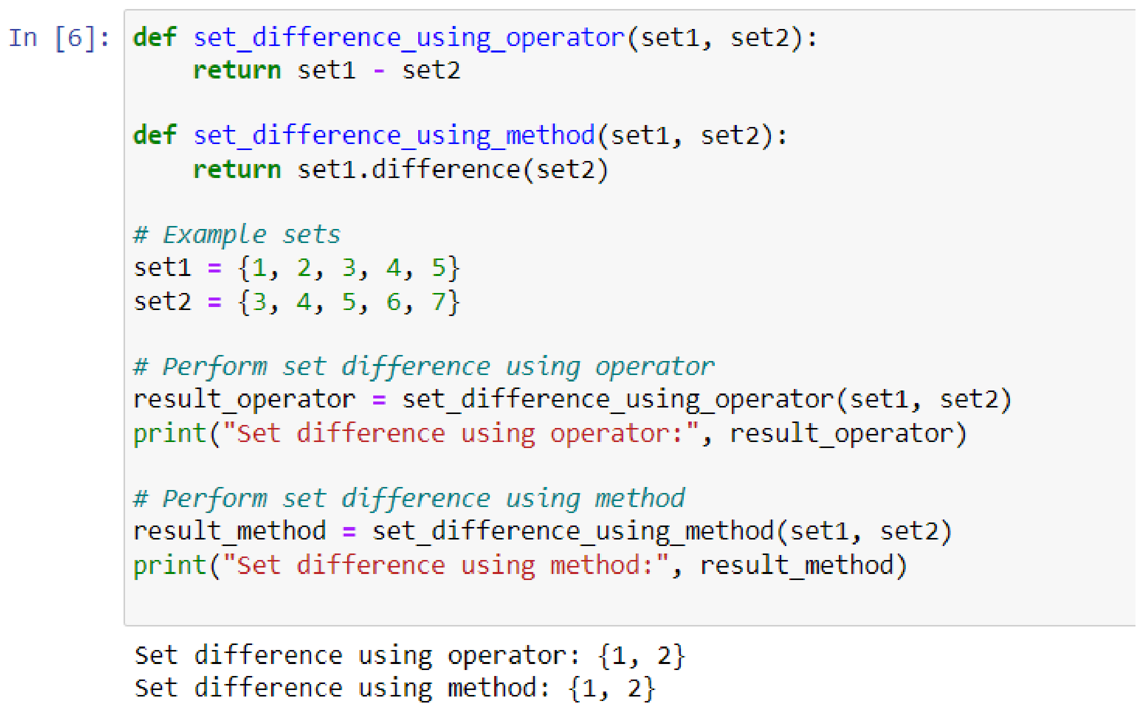
In this example, the lines of code within the if and else blocks are indented to indicate that they are part of those blocks. The nested if statement is further indented to show that it is within the outer if block.

6. Write a program to perform set difference operation.

Program:

```
def set_difference_using_operator(set1, set2):
    return set1 - set2
def set_difference_using_method(set1, set2):
    return set1.difference(set2)
# Example sets
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}
# Perform set difference using operator
result_operator = set_difference_using_operator(set1, set2)
print("Set difference using operator:", result_operator)
# Perform set difference using method
result_method = set_difference_using_method(set1, set2)
print("Set difference using method:", result_method)
```

Output:



```
In [6]: def set_difference_using_operator(set1, set2):
            return set1 - set2

        def set_difference_using_method(set1, set2):
            return set1.difference(set2)

        # Example sets
        set1 = {1, 2, 3, 4, 5}
        set2 = {3, 4, 5, 6, 7}

        # Perform set difference using operator
        result_operator = set_difference_using_operator(set1, set2)
        print("Set difference using operator:", result_operator)

        # Perform set difference using method
        result_method = set_difference_using_method(set1, set2)
        print("Set difference using method:", result_method)
```

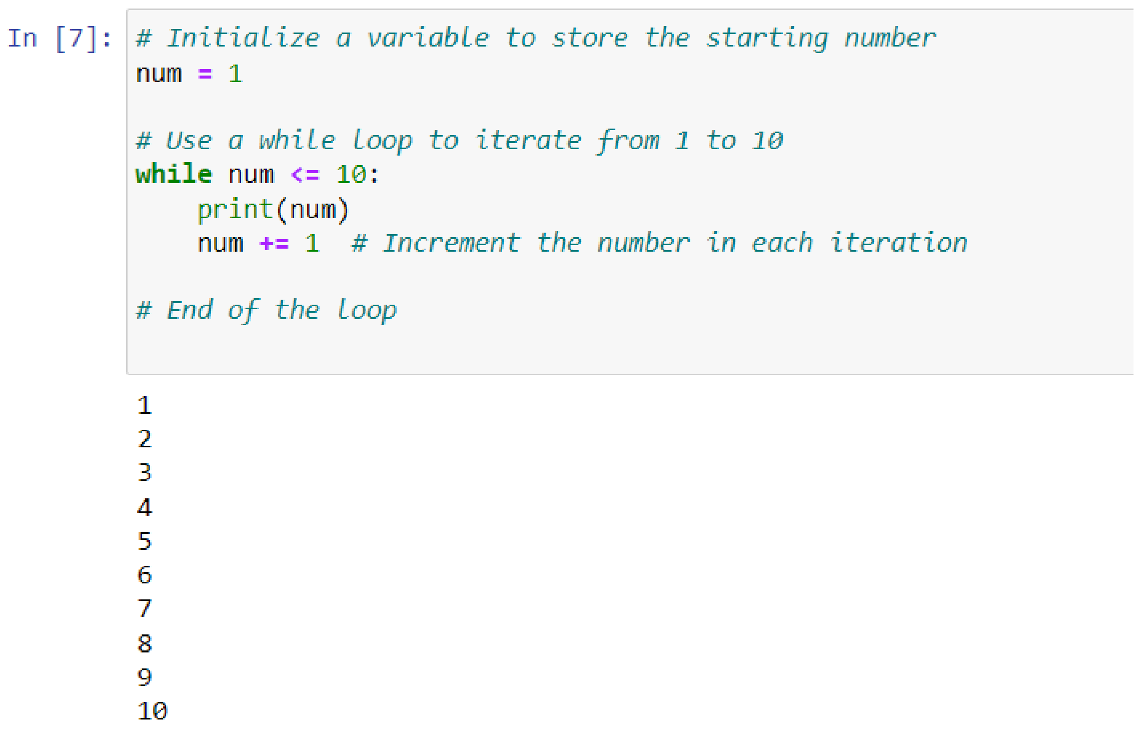
```
Set difference using operator: {1, 2}
Set difference using method: {1, 2}
```

7. Write a Python program to print numbers from 1 to 10 using a while loop.

Program:

```
# Initialize a variable to store the starting number

num = 1

# Use a while loop to iterate from 1 to 10

while num <= 10:

    print(num)

    num += 1  # Increment the number in each iteration

# End of the loop
```

Output:



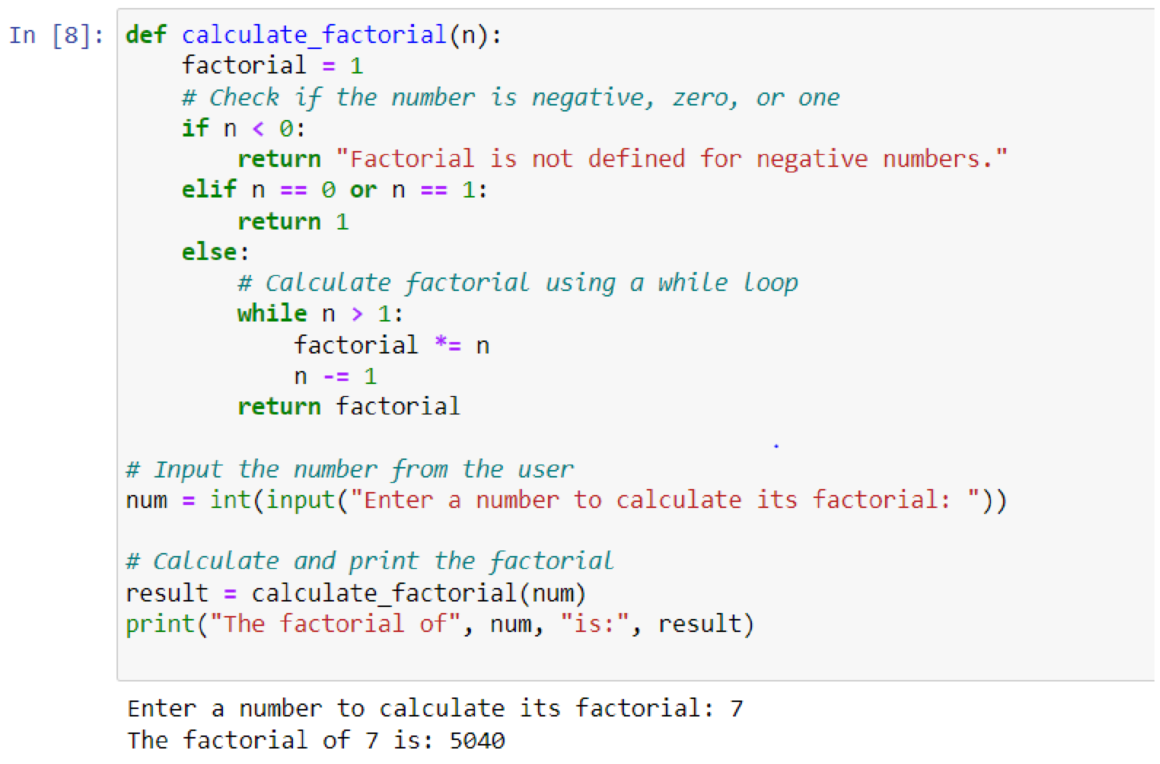
```
In [7]: # Initialize a variable to store the starting number
        num = 1

        # Use a while loop to iterate from 1 to 10
        while num <= 10:
            print(num)
            num += 1  # Increment the number in each iteration

        # End of the loop

        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
```

8. Write a program to calculate the factorial of a number using a while loop.

Program:

```
def calculate_factorial(n):
    factorial = 1
    # Check if the number is negative, zero, or one
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0 or n == 1:
        return 1
    else:
        # Calculate factorial using a while loop
        while n > 1:
            factorial *= n
            n -= 1
        return factorial

# Input the number from the user

num = int(input("Enter a number to calculate its factorial: "))

# Calculate and print the factorial

result = calculate_factorial(num)

print("The factorial of", num, "is:", result)
```

Output:



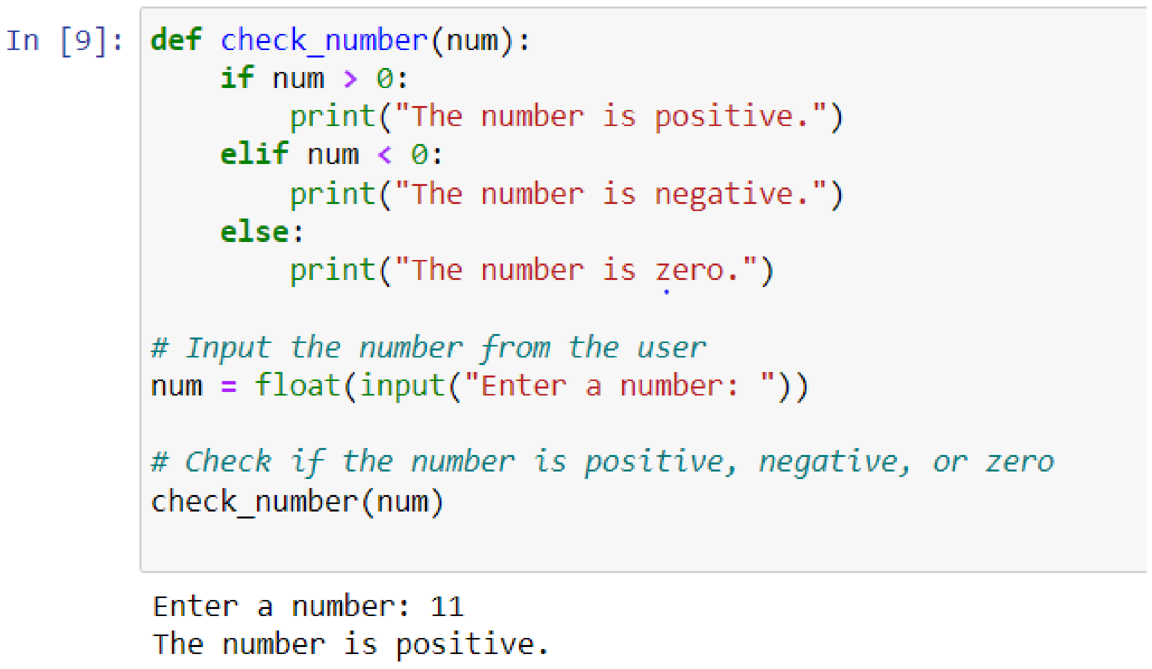
```
In [8]: def calculate_factorial(n):
            factorial = 1
            # Check if the number is negative, zero, or one
            if n < 0:
                return "Factorial is not defined for negative numbers."
            elif n == 0 or n == 1:
                return 1
            else:
                # Calculate factorial using a while loop
                while n > 1:
                    factorial *= n
                    n -= 1
                return factorial

        # Input the number from the user
        num = int(input("Enter a number to calculate its factorial: "))

        # Calculate and print the factorial
        result = calculate_factorial(num)
        print("The factorial of", num, "is:", result)

        Enter a number to calculate its factorial: 7
        The factorial of 7 is: 5040
```

9. Write a Python program to check if a number is positive, negative, or zero using if-elif-else statements.

Program:

```
def check_number(num):

    if num > 0:

        print("The number is positive.")

    elif num < 0:

        print("The number is negative.")
    else:
        print("The number is zero.")
# Input the number from the user
num = float(input("Enter a number: "))
# Check if the number is positive, negative, or zero
check_number(num)
```

Output:



```
In [9]: def check_number(num):
            if num > 0:
                print("The number is positive.")
            elif num < 0:
                print("The number is negative.")
            else:
                print("The number is zero.")

        # Input the number from the user
        num = float(input("Enter a number: "))

        # Check if the number is positive, negative, or zero
        check_number(num)
```

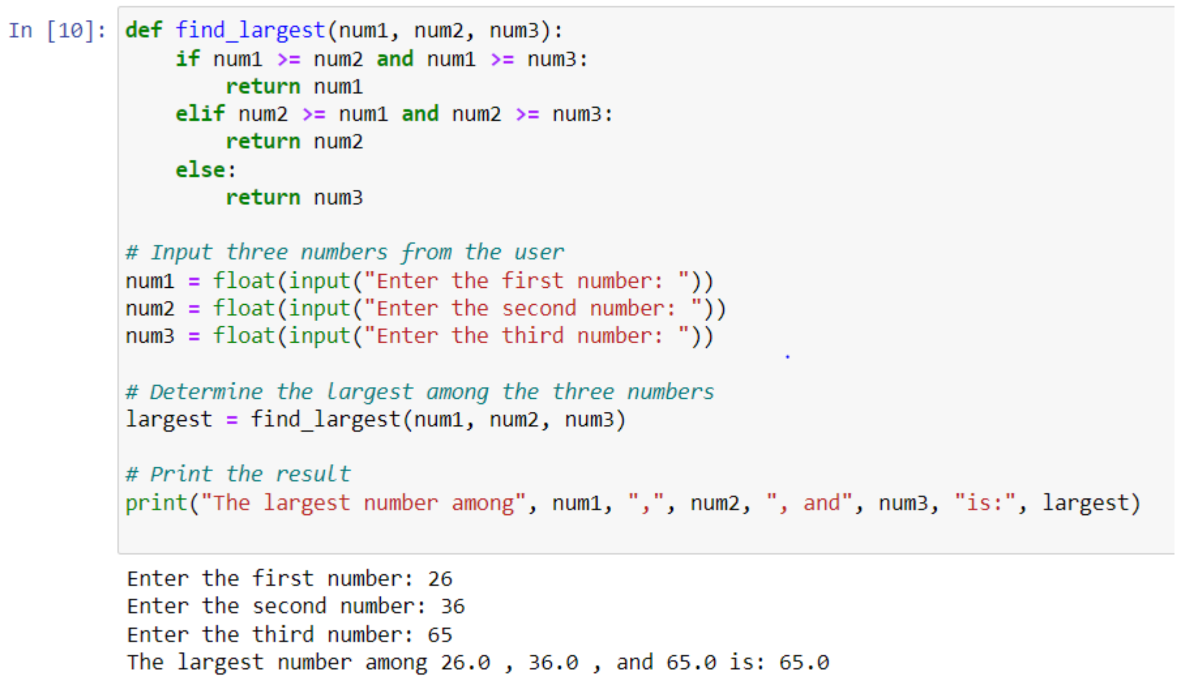
```
Enter a number: 11
The number is positive.
```

10. Write program to determine the largest among three numbers using conditional statements.

Program:

```
def find_largest(num1, num2, num3):
    if num1 >= num2 and num1 >= num3:
        return num1
    elif num2 >= num1 and num2 >= num3:
        return num2
    else:
        return num3
# Input three numbers from the user
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
num3 = float(input("Enter the third number: "))
# Determine the largest among the three numbers
largest = find_largest(num1, num2, num3)
# Print the result
print("The largest number among", num1, ",", num2, ", and", num3, "is:", largest)
```

Output:



```
In [10]: def find_largest(num1, num2, num3):
             if num1 >= num2 and num1 >= num3:
                 return num1
             elif num2 >= num1 and num2 >= num3:
                 return num2
             else:
                 return num3

         # Input three numbers from the user
         num1 = float(input("Enter the first number: "))
         num2 = float(input("Enter the second number: "))
         num3 = float(input("Enter the third number: "))

         # Determine the largest among the three numbers
         largest = find_largest(num1, num2, num3)

         # Print the result
         print("The largest number among", num1, ",", num2, ", and", num3, "is:", largest)

         Enter the first number: 26
         Enter the second number: 36
         Enter the third number: 65
         The largest number among 26.0 , 36.0 , and 65.0 is: 65.0
```

11. Write a Python program to create a numpy array filled with ones of given shape.

Program:

```
import numpy as np
def create_ones_array(shape):
    # Create a NumPy array filled with ones of the given shape
    ones_array = np.ones(shape)
```

    return ones_array

Input the shape of the array from the user

rows = int(input("Enter the number of rows: "))

columns = int(input("Enter the number of columns: "))

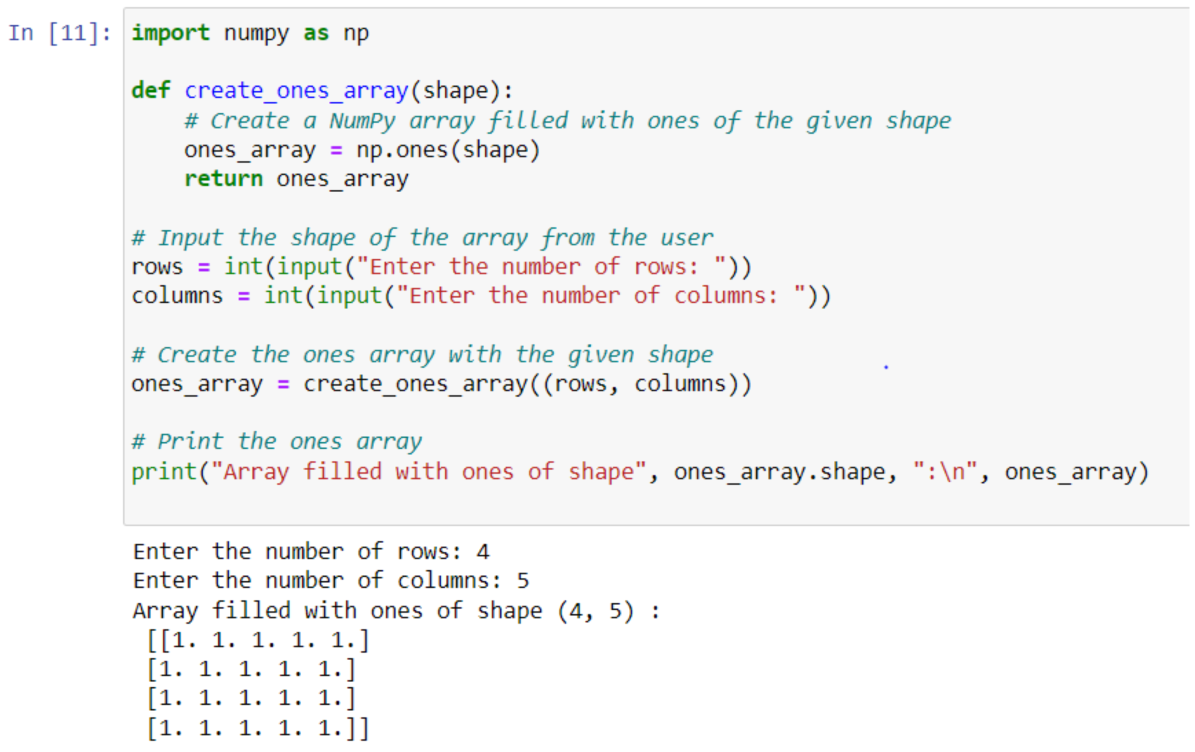
Create the ones array with the given shape

ones_array = create_ones_array((rows, columns))

Print the ones array

print("Array filled with ones of shape", ones_array.shape, ":\n", ones_array)

Output:



```
In [11]: import numpy as np

         def create_ones_array(shape):
             # Create a NumPy array filled with ones of the given shape
             ones_array = np.ones(shape)
             return ones_array

         # Input the shape of the array from the user
         rows = int(input("Enter the number of rows: "))
         columns = int(input("Enter the number of columns: "))

         # Create the ones array with the given shape
         ones_array = create_ones_array((rows, columns))

         # Print the ones array
         print("Array filled with ones of shape", ones_array.shape, ":\n", ones_array)

         Enter the number of rows: 4
         Enter the number of columns: 5
         Array filled with ones of shape (4, 5) :
          [[1. 1. 1. 1. 1.]
          [1. 1. 1. 1. 1.]
          [1. 1. 1. 1. 1.]
          [1. 1. 1. 1. 1.]]
```

12. Write a program to create a 2D numpy array initialized with random integers.

Program:

import numpy as np

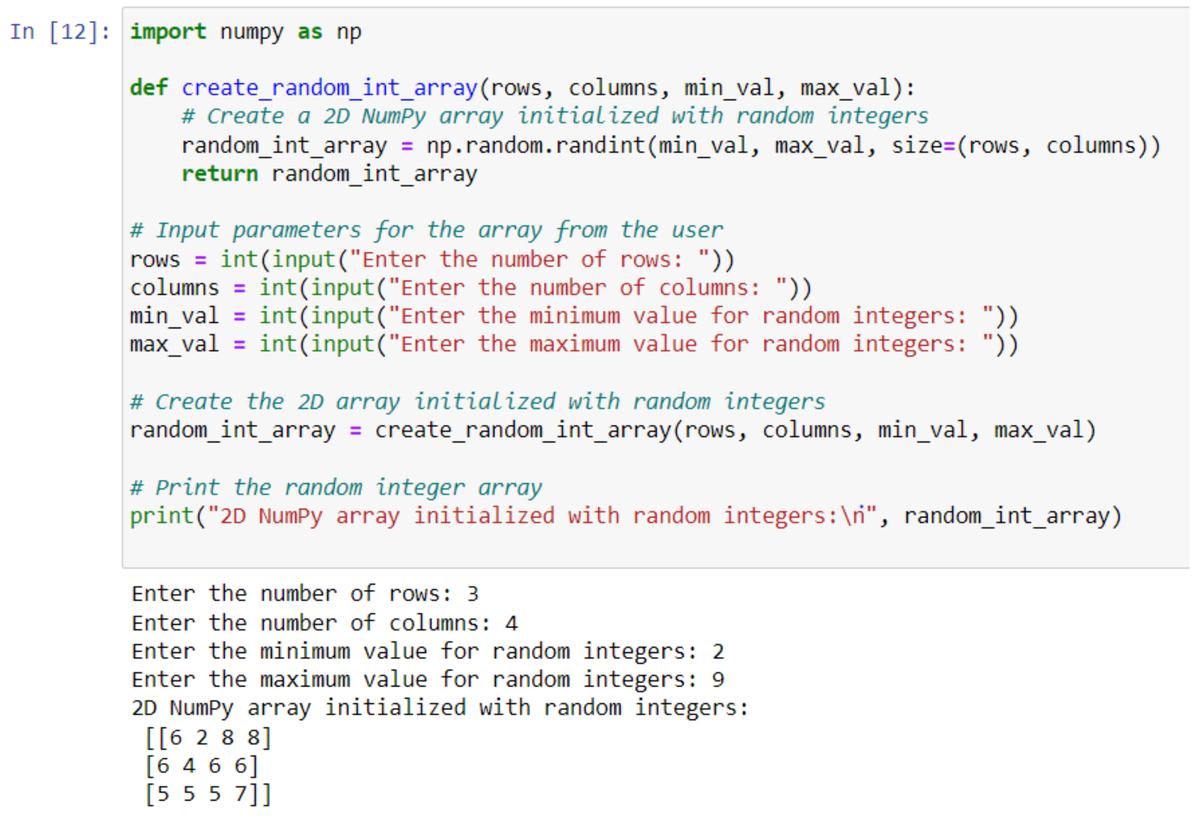
def create_random_int_array(rows, columns, min_val, max_val):

    # Create a 2D NumPy array initialized with random integers

    random_int_array = np.random.randint(min_val, max_val, size=(rows, columns))

```
    return random_int_array

# Input parameters for the array from the user

rows = int(input("Enter the number of rows: "))

columns = int(input("Enter the number of columns: "))

min_val = int(input("Enter the minimum value for random integers: "))

max_val = int(input("Enter the maximum value for random integers: "))

# Create the 2D array initialized with random integers

random_int_array = create_random_int_array(rows, columns, min_val, max_val)

# Print the random integer array

print("2D NumPy array initialized with random integers:\n", random_int_array)
```

Output:



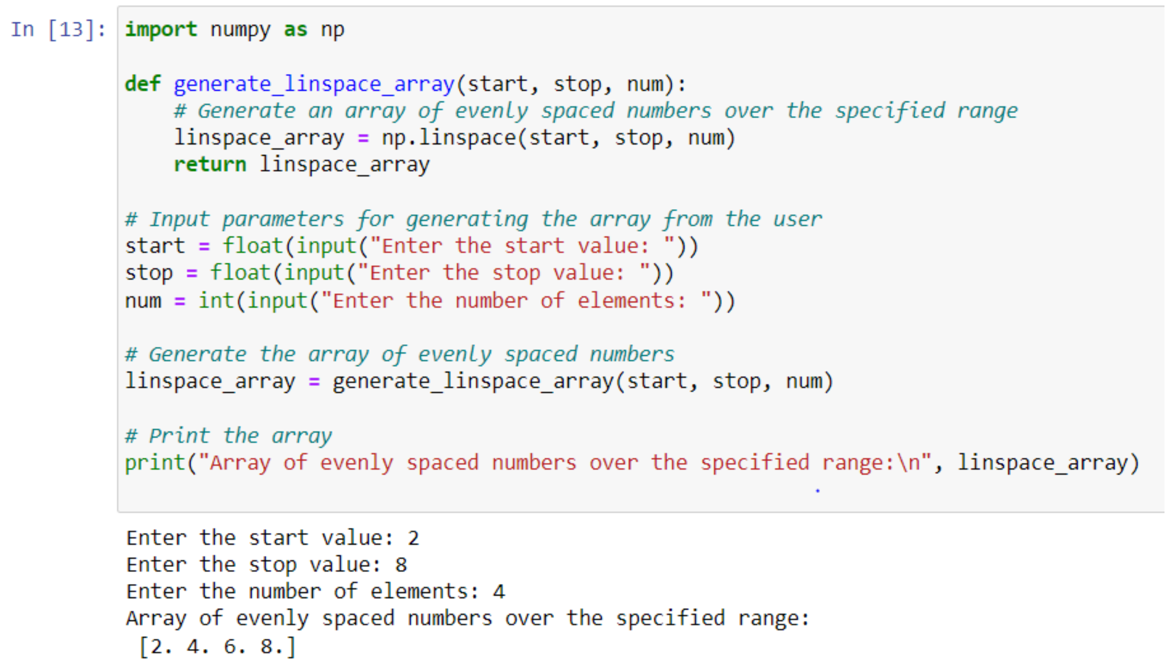
```
In [12]: import numpy as np

         def create_random_int_array(rows, columns, min_val, max_val):
             # Create a 2D NumPy array initialized with random integers
             random_int_array = np.random.randint(min_val, max_val, size=(rows, columns))
             return random_int_array

         # Input parameters for the array from the user
         rows = int(input("Enter the number of rows: "))
         columns = int(input("Enter the number of columns: "))
         min_val = int(input("Enter the minimum value for random integers: "))
         max_val = int(input("Enter the maximum value for random integers: "))

         # Create the 2D array initialized with random integers
         random_int_array = create_random_int_array(rows, columns, min_val, max_val)

         # Print the random integer array
         print("2D NumPy array initialized with random integers:\n", random_int_array)

         Enter the number of rows: 3
         Enter the number of columns: 4
         Enter the minimum value for random integers: 2
         Enter the maximum value for random integers: 9
         2D NumPy array initialized with random integers:
          [[6 2 8 8]
          [6 4 6 6]
          [5 5 5 7]]
```

13. Write a Python program to generate an array of evenly spaced numbers over a specified range using linspace.

Program:

```
import numpy as np

def generate_linspace_array(start, stop, num):

  # Generate an array of evenly spaced numbers over the specified range

  linspace_array = np.linspace(start, stop, num)

  return linspace_array

# Input parameters for generating the array from the user

start = float(input("Enter the start value: "))

stop = float(input("Enter the stop value: "))

num = int(input("Enter the number of elements: "))

# Generate the array of evenly spaced numbers

linspace_array = generate_linspace_array(start, stop, num)

# Print the array

print("Array of evenly spaced numbers over the specified range:\n", linspace_array)
```

Output:



```
In [13]: import numpy as np

         def generate_linspace_array(start, stop, num):
             # Generate an array of evenly spaced numbers over the specified range
             linspace_array = np.linspace(start, stop, num)
             return linspace_array

         # Input parameters for generating the array from the user
         start = float(input("Enter the start value: "))
         stop = float(input("Enter the stop value: "))
         num = int(input("Enter the number of elements: "))

         # Generate the array of evenly spaced numbers
         linspace_array = generate_linspace_array(start, stop, num)

         # Print the array
         print("Array of evenly spaced numbers over the specified range:\n", linspace_array)

         Enter the start value: 2
         Enter the stop value: 8
         Enter the number of elements: 4
         Array of evenly spaced numbers over the specified range:
          [2. 4. 6. 8.]
```

14. Write a program to generate an array of 10 equally spaced values between 1 and 100 using linspace.

Program:

import numpy as np

Generate an array of 10 equally spaced values between 1 and 100

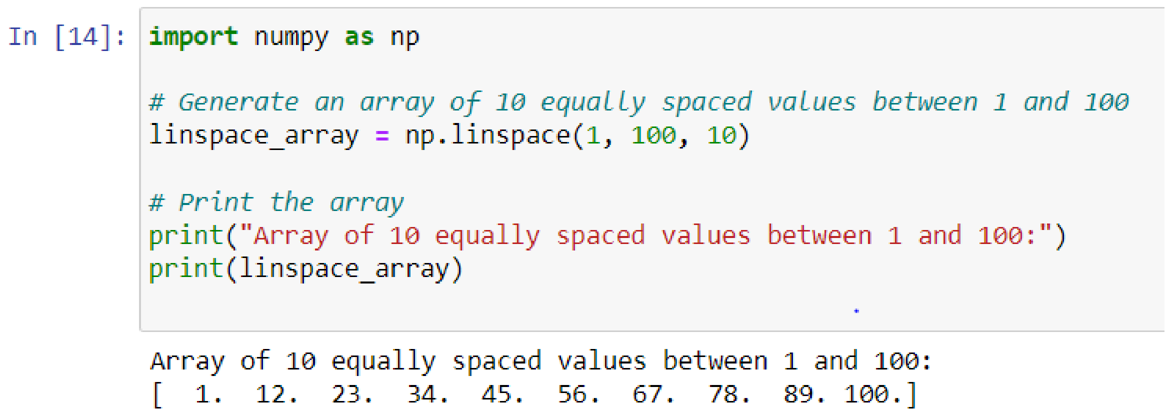
linspace_array = np.linspace(1, 100, 10)

# Print the array

print("Array of 10 equally spaced values between 1 and 100:")

print(linspace_array)

Output:



```
In [14]: import numpy as np

         # Generate an array of 10 equally spaced values between 1 and 100
         linspace_array = np.linspace(1, 100, 10)

         # Print the array
         print("Array of 10 equally spaced values between 1 and 100:")
         print(linspace_array)

         Array of 10 equally spaced values between 1 and 100:
         [  1.  12.  23.  34.  45.  56.  67.  78.  89. 100.]
```

15. Write a Python program to create an array containing even numbers from 2 to 20 using arrange.

Program:

import numpy as np

# Create an array containing even numbers from 2 to 20 using arange

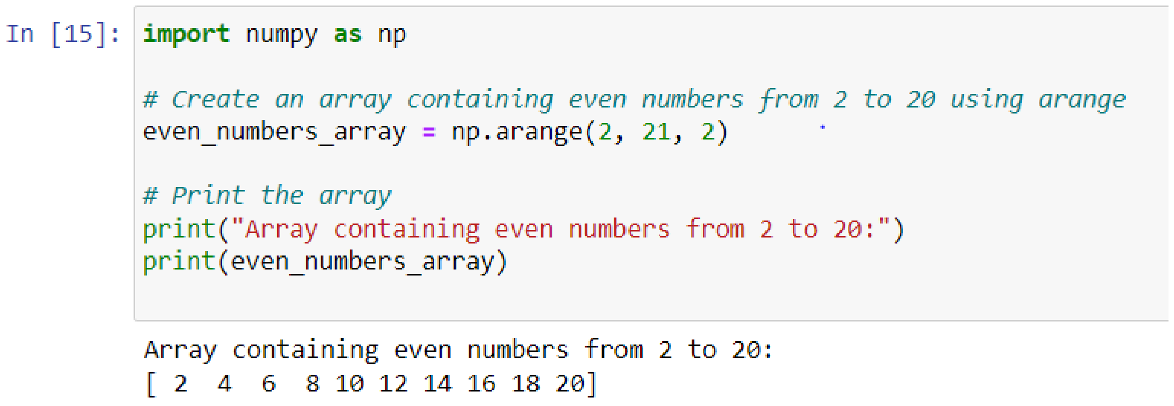
even_numbers_array = np.arange(2, 21, 2)

# Print the array

print("Array containing even numbers from 2 to 20:")

print(even_numbers_array)

Output:



```
In [15]: import numpy as np

         # Create an array containing even numbers from 2 to 20 using arange
         even_numbers_array = np.arange(2, 21, 2)

         # Print the array
         print("Array containing even numbers from 2 to 20:")
         print(even_numbers_array)

         Array containing even numbers from 2 to 20:
         [ 2  4  6  8 10 12 14 16 18 20]
```

16. Write a program to create an array containing numbers from 1 to 10 with a step size of 0.5 using arange.

Program:

import numpy as np

Create an array containing numbers from 1 to 10 with a step size of 0.5 using arange

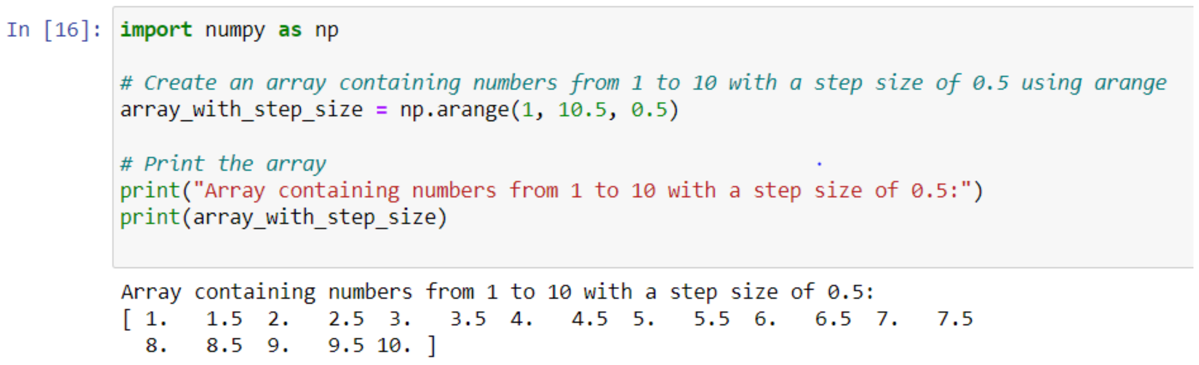
array_with_step_size = np.arange(1, 10.5, 0.5)

Print the array

print("Array containing numbers from 1 to 10 with a step size of 0.5:")

print(array_with_step_size)

Output:



```
In [16]: import numpy as np

         # Create an array containing numbers from 1 to 10 with a step size of 0.5 using arange
         array_with_step_size = np.arange(1, 10.5, 0.5)

         # Print the array
         print("Array containing numbers from 1 to 10 with a step size of 0.5:")
         print(array_with_step_size)

         Array containing numbers from 1 to 10 with a step size of 0.5:
         [ 1.   1.5  2.   2.5  3.   3.5  4.   4.5  5.   5.5  6.   6.5  7.   7.5
           8.   8.5  9.   9.5 10. ]
```