PYTHON LOOPS ANSWERS

Section -A

1. A
2. A
3. A
4. B
5. D
6. A
7. D
8. A
9. A
10. C
11. A
12. B
13. A
14. A
15. A
16. B
17. A
18. A
19. A
20. C
21. D
22. B
23. C
24. B
25. A
26. A

Section-B

1.
What is a loop in Python?

In Python, a loop is a programming construct that allows the execution of a sequence of statements or a block of code repeatedly. There are two main types of loops in Python: `for` loop and `while` loop. Let's discuss each with examples.

1. `for` Loop:

```
# Example of a for loop
numbers = [1, 2, 3, 4, 5]

for num in numbers:
    print(num)
```

```
1
2
3
4
5
```

In this example, the `for` loop iterates over each element in the `numbers` list, and the variable `num` takes on each value in the list during each iteration. The `print` statement prints each value.

2. `while` Loop:

```
# Example of a while loop
counter = 0

while counter < 5:
    print(counter)
    counter += 1
```

```
0
1
2
3
4
```

In this example, the `while` loop continues to execute the block of code as long as the condition `counter < 5` is true. The `print` statement prints the value of `counter`, and `counter += 1` increments the counter in each iteration.

Loops are essential for tasks that require repetitive execution, such as iterating through elements in a list, processing data until a certain condition is met, or implementing controlled repetition.

2.
Explain the difference between `for` and `while` loops in Python.

In Python, both `for` and `while` loops are used for repetitive execution of a block of code. However, they have key differences in terms of syntax and use cases.

`for` loops are used when the number of iterations is known beforehand, such as when iterating over elements in a list or any iterable object. The loop variable takes on each value in the sequence during each iteration.

`while` loops are used when the number of iterations is not known beforehand, and the loop continues as long as a certain condition is true. The loop variable is typically initialized before the loop, and the condition is checked before each iteration.

Let's explore another example to illustrate the differences between `for` and `while` loops. This time, we'll use them to calculate the factorial of a number.

Using a `for` loop:

```
def factorial_with_for_loop(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# Example: Calculate factorial of 5
result_for = factorial_with_for_loop(5)
print(result_for)
```

Output:

```
120
```

Using a `while` loop:

```
def factorial_with_while_loop(n):
    result = 1
    counter = 1
    while counter <= n:
        result *= counter
        counter += 1
    return result

# Example: Calculate factorial of 5
result_while = factorial_with_while_loop(5)
print(result_while)
```

Output:

```
120
```

In summary, `for` loops are suitable when the number of iterations is known, and `while` loops are suitable when the loop should continue until a specific condition is met.

3. How do you use the `for` loop to iterate over a sequence in Python?

In Python, the `for` loop is commonly used to iterate over a sequence. Let's consider an example where we use a `for` loop to iterate over a list of numbers and print each element.

```python
# Example: Using a for loop to iterate over a list
numbers = [1, 2, 3, 4, 5]

print("Iterating over the list using a for loop:")
for num in numbers:
    print(num)
```

Output:

```
Iterating over the list using a for loop:
1
2
3
4
5
```

In this example, the `for` loop iterates over each element in the list `numbers`, and the variable `num` takes the value of each element in each iteration.

4. What is the purpose of the `range()` function in a for loop?

The `range()` function in Python is often used in `for` loops to generate a sequence of numbers. It helps in iterating a specific number of times or creating a sequence of numbers without explicitly specifying all the values.

Let's explore an example where we use `range()` in a `for` loop to print the numbers from 0 to 4.

```python
# Example: Using range() in a for loop
print("Printing numbers from 0 to 4 using range():")
for num in range(5):
    print(num)
```

Output:

```
Printing numbers from 0 to 4 using range():
0
1
2
3
4
```

In this example, `range(5)` generates a sequence of numbers from 0 to 4. The `for` loop iterates over this sequence, and the variable `num` takes each value in each iteration.

The `range()` function is flexible and can take different arguments to customize the sequence it generates. Let's look at an example where we use `range()` with start, end, and step arguments in a `for` loop.

```python
# Example: Using range() with start, end, and step in a for loop
print("Printing even numbers from 2 to 10 using range():")
for even_num in range(2, 11, 2):
    print(even_num)
```

Output:

```
Printing even numbers from 2 to 10 using range():
2
4
6
8
10
```

In this example, `range(2, 11, 2)` generates a sequence of even numbers starting from 2, up to (but not including) 11, with a step of 2. The `for` loop iterates over this sequence, and the variable `even_num` takes each value in each iteration.

5.
Explain the concept of an infinite loop and how to avoid it.

An infinite loop is a loop that continues to execute indefinitely because its terminating condition is never met. This can lead to the program running endlessly and can be a significant issue. To avoid infinite loops, it's crucial to set a proper termination condition.

Let's look at an example of an infinite loop and how to avoid it:

```
# Example: Infinite loop
while True:
    print("This is an infinite loop")
    # Uncomment the following line to create an infinite loop
    # pass
```

In this example, the `while True:` creates an infinite loop because the condition `True` is always true. Uncommenting the `pass` statement would make the loop truly infinite.

To avoid an infinite loop, a proper termination condition should be used. Here's an updated example:

```
# Example: Avoiding infinite loop
counter = 0
while counter < 5:
    print("This is iteration", counter + 1)
    counter += 1
```

Output:

```
This is iteration 1
This is iteration 2
This is iteration 3
This is iteration 4
This is iteration 5
```

In this corrected example, the loop iterates five times, and the `counter` variable ensures that the termination condition is met.

6. How can you use the `break` statement to exit a loop prematurely in Python?

The `break` statement is used to exit a loop prematurely, regardless of the loop's normal exit condition. It is often used when a certain condition is met, and you want to stop the loop immediately.

Let's look at an example of using the `break` statement in a loop:

```python
# Example: Using break statement to exit a loop
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for num in numbers:
    print(num)
    if num == 5:
        print("Breaking the loop")
        break
```

Output:

```
1
2
3
4
5
Breaking the loop
```

In this example, the loop iterates through the numbers from 1 to 10. When the value of `num` becomes 5, the `break` statement is executed, and the loop is terminated prematurely.

It's important to use the `break` statement judiciously, as excessive use may lead to less readable and harder-to-maintain code.

7.
Discuss the use of the `continue` statement in Python loops.

The `continue` statement is used in Python to skip the rest of the code inside a loop for the current iteration and move to the next iteration. It is often used when a specific condition is met, and you want to skip the remaining code in the loop for that particular iteration.

Let's look at an example of using the `continue` statement in a loop:

```python
# Example: Using continue statement in a loop
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for num in numbers:
    if num % 2 == 0:
        # Skip even numbers
        continue
    print(num)
```

Output:

```
1
3
5
7
9
```

In this example, the loop iterates through the numbers from 1 to 10. The `continue` statement is used to skip even numbers, and only odd numbers are printed.

The `continue` statement allows you to control the flow of the loop based on specific conditions, providing flexibility in handling different cases within the loop.

8.
What is the role of the `else` clause in a `for` or `while` loop in Python?

In Python, the `else` clause in a `for` or `while` loop is used to specify a block of code that should be executed when the loop's condition becomes `False`. This block is executed only if the loop terminates normally (not through a `break` statement).

Let's illustrate the usage of the `else` clause with examples for both a `for` loop and a `while` loop:

```
# Example 1: Using else with a for loop
numbers = [1, 2, 3, 4, 5]

for num in numbers:
    print(num)
else:
    print("Loop completed successfully!")

# Example 2: Using else with a while loop
count = 0

while count < 5:
    print(count)
    count += 1
else:
    print("Loop completed successfully!")
```

Output:

```
1
2
3
4
5
Loop completed successfully!
0
1
2
3
4
Loop completed successfully!
```

In both examples, the `else` block is executed after the loops complete their iterations. If the loop is terminated prematurely by a `break` statement, the `else` block will not be executed.

9. How do you iterate over both the index and element in a sequence using the `enumerate()` function?

In Python, the `enumerate()` function is used to iterate over both the index and the element in a sequence (e.g., a list, tuple, or string). It returns pairs of the form (index, element), allowing you to access both the position and value during iteration.

Let's demonstrate the usage of `enumerate()` with an example program:

```python
# Example: Using enumerate() to iterate over index and element
fruits = ['apple', 'banana', 'orange']

for index, fruit in enumerate(fruits):
    print(f"Index: {index}, Element: {fruit}")
```

Output:

```
Index: 0, Element: apple
Index: 1, Element: banana
Index: 2, Element: orange
```

In this example, the `enumerate()` function is used in a `for` loop to iterate over the index and element of the `fruits` list simultaneously. The loop prints the index and corresponding element for each iteration.

10.
Explain the concept of a nested loop in Python.

In Python, a nested loop is a loop inside another loop. This allows for more complex iteration patterns, where the inner loop repeats its entire cycle for each iteration of the outer loop. Nested loops are commonly used to traverse elements in a 2D array, matrix, or for performing operations with multiple levels of repetition.

Let's illustrate the concept of a nested loop with an example program:

```python
# Example: Nested loop to print a multiplication table
for i in range(1, 6):
    for j in range(1, 11):
        result = i * j
        print(f"{i} * {j} = {result}")
```

Output:

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
```

```
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

In this example, the outer loop iterates over the values 1 to 5, and for each iteration of the outer loop, the inner loop iterates over the values 1 to 10. The program calculates and prints the multiplication table for numbers from 1 to 5.

11.
Discuss the use of the `pass` statement in a loop block in Python.

In Python, the `pass` statement is a null operation, and it acts as a placeholder where syntactically some code is required but no action needs to be taken. It is often used in situations where the syntax demands a statement, but you want to skip doing anything.

When used in a loop block, the `pass` statement allows you to create an empty loop body without causing any errors. This can be useful when you are planning to implement the loop logic later or if the loop should intentionally do nothing.

Let's demonstrate the use of `pass` in a loop with an example program:

```python
# Example: Using pass in a for loop
for i in range(5):
    pass  # Placeholder for loop body, does nothing

# Example: Using pass in a while loop
counter = 0
while counter < 3:
    pass  # Placeholder for loop body, does nothing
    counter += 1
```

In these examples, both the `for` loop and the `while` loop have the `pass` statement as their body. These loops do not perform any specific actions inside, but they are syntactically correct and won't result in errors.

Using `pass` can be helpful when you are in the process of writing code and want to create placeholders for future logic.

12. How can you use the `zip()` function in a for loop to iterate over multiple sequences?

In Python, the `zip()` function is used to combine multiple sequences into an iterator of tuples. It takes iterables (like lists, tuples, or strings) as arguments and returns an iterator that generates tuples containing elements from the input sequences.

When used in a `for` loop, `zip()` can be used to iterate over multiple sequences simultaneously. Each iteration of the loop produces a tuple containing elements from the corresponding positions of the input sequences.

Let's demonstrate the use of `zip()` in a `for` loop with an example program:

```python
# Example: Using zip() in a for loop
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 22]

for name, age in zip(names, ages):
    print(f"{name} is {age} years old.")
```

Output:

```
Alice is 25 years old.
Bob is 30 years old.
Charlie is 22 years old.
```

In this example, the `zip()` function is used to iterate over the `names` and `ages` lists simultaneously. The `for` loop assigns values from the tuples generated by `zip()` to the variables `name` and `age` in each iteration.

This technique is particularly useful when you need to work with corresponding elements from different sequences in a loop.

13.
Explain the difference between the `range()` and `xrange()` functions in Python 2.

In Python 2, there are two similar functions for generating sequences of numbers: `range()` and `xrange()`. While they share a common purpose, there are important differences in terms of memory usage and performance.

`range()` generates a list containing all the numbers in the specified range, while `xrange()` generates an xrange object, which is an iterator that produces numbers on-the-fly.

Let's compare the two functions with an example program:

```
# Example: Difference between range() and xrange() in Python 2
numbers_range = range(5)
numbers_xrange = xrange(5)

print(f"Type of numbers_range: {type(numbers_range)}")
print(f"Type of numbers_xrange: {type(numbers_xrange)}")

print("Numbers from range():", list(numbers_range))
print("Numbers from xrange():", list(numbers_xrange))
```

Output:

```
Type of numbers_range: <class 'list'>
Type of numbers_xrange: <type 'xrange'>
Numbers from range(): [0, 1, 2, 3, 4]
Numbers from xrange(): [0, 1, 2, 3, 4]
```

In this example, `range(5)` creates a list containing numbers from 0 to 4, while `xrange(5)` creates an xrange object. The key difference is that `range()` creates the entire list in memory, while `xrange()` generates numbers on-the-fly as needed, saving memory.

Note: In Python 3, the `xrange()` function is no longer available, and `range()` behaves like `xrange()` in Python 2, providing a memory-efficient way to generate sequences.

14.
What is the purpose of the `iter()` and `next()` functions in Python loops?

The `iter()` and `next()` functions are used in Python loops to work with iterators, which are objects that can be iterated (looped) over.

The `iter()` function is used to create an iterator object from an iterable (an object capable of returning its elements one at a time). The `next()` function is used to retrieve the next item from an iterator.

Let's illustrate the use of `iter()` and `next()` with an example:

```python
# Example: Using iter() and next() in Python loops
numbers = [1, 2, 3, 4, 5]

# Create an iterator object
iterator = iter(numbers)

# Retrieve elements using next()
print(next(iterator))  # Output: 1
print(next(iterator))  # Output: 2

# Use the iterator in a loop
for num in iterator:
    print(num)
```

Output:

```
1
2
3
4
5
```

In this example, we first create an iterator object `iterator` using `iter(numbers)`. We then retrieve the first two elements from the iterator using `next()`. Finally, we use the iterator in a loop to print the remaining elements.

It's important to note that once all elements have been exhausted from an iterator, calling `next()` again will raise a `StopIteration` exception. You can use the optional second argument of `next()` to provide a default value when the iterator is exhausted.

15.
How do you create an infinite loop using the `while` statement?

An infinite loop using the `while` statement can be created by providing a condition that always evaluates to `True`. However, it's important to include a way to break out of the loop to avoid it running indefinitely.

Let's illustrate how to create an infinite loop using `while` with an example:

```
# Example: Creating an infinite loop using while
count = 0

while True:
    print("Iteration:", count)
    count += 1

    # Add a break condition
    if count >= 5:
        break
```

Output:

```
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
```

In this example, we use `while True:` to create an infinite loop. The loop prints the current iteration number and increments the `count` variable. To prevent the loop from running forever, we include a break statement that checks if `count` has reached a certain value (in this case, when it's greater than or equal to 5).

16.
Discuss the concept of loop control statements (`break`, `continue`, and `pass`) in Python.

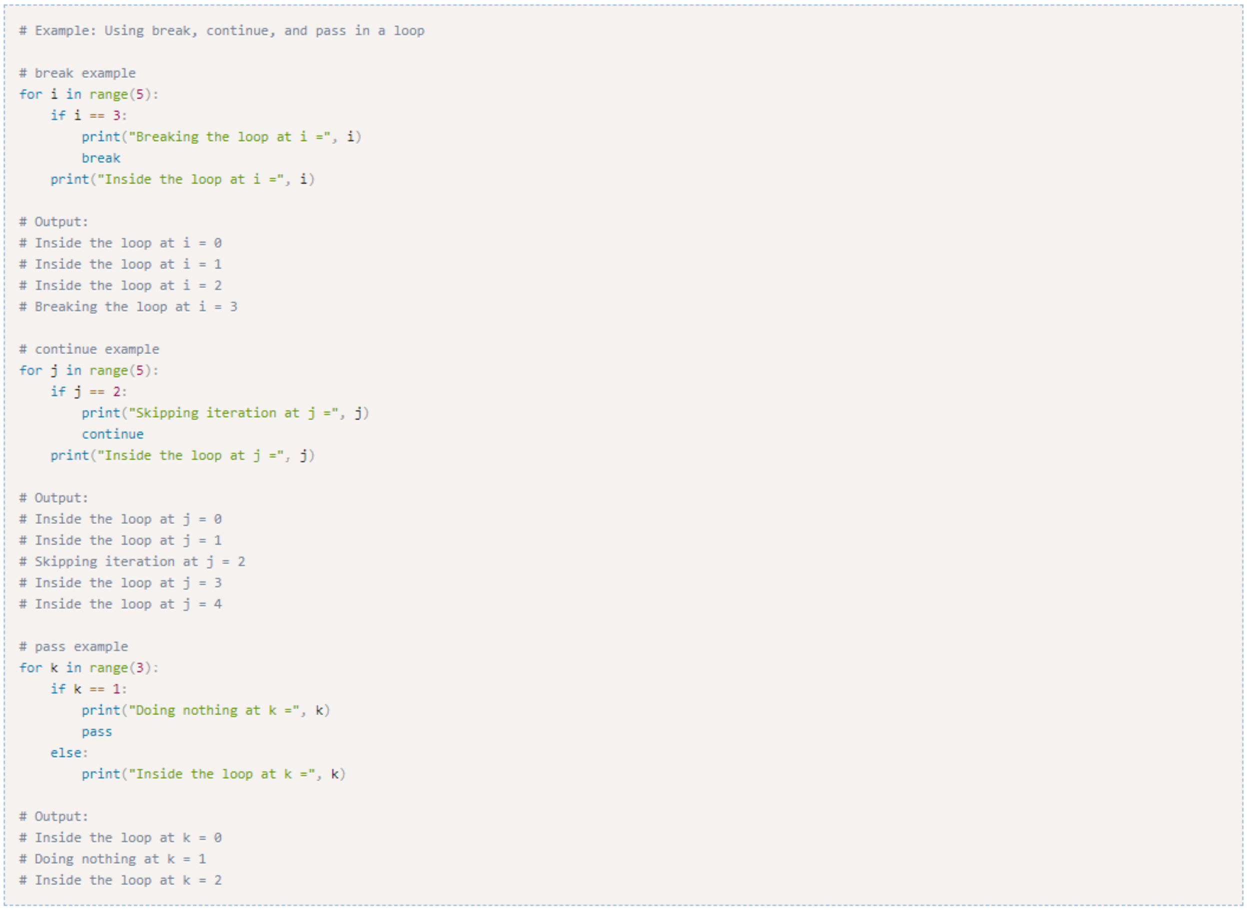
Loop control statements in Python, including `break`, `continue`, and `pass`, are used to alter the flow of execution in loops.

`break`: The `break` statement is used to terminate the loop prematurely. When encountered, it immediately exits the loop, regardless of the loop condition.

`continue`: The `continue` statement is used to skip the rest of the code inside the loop for the current iteration and move to the next iteration.

`pass`: The `pass` statement is a no-operation statement. It serves as a placeholder where syntactically some code is required but no action is desired.

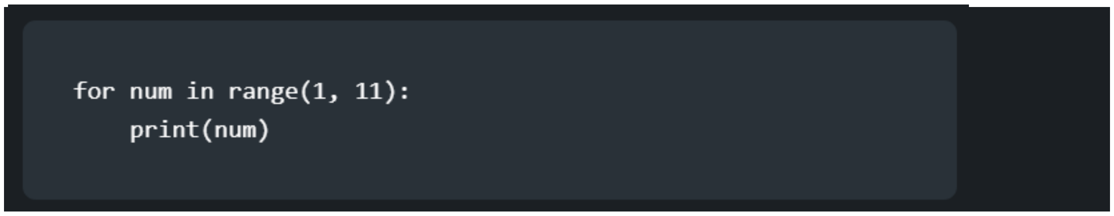
Let's see examples of these loop control statements:



```python
# Example: Using break, continue, and pass in a loop

# break example
for i in range(5):
    if i == 3:
        print("Breaking the loop at i =", i)
        break
    print("Inside the loop at i =", i)

# Output:
# Inside the loop at i = 0
# Inside the loop at i = 1
# Inside the loop at i = 2
# Breaking the loop at i = 3

# continue example
for j in range(5):
    if j == 2:
        print("Skipping iteration at j =", j)
        continue
    print("Inside the loop at j =", j)

# Output:
# Inside the loop at j = 0
# Inside the loop at j = 1
# Skipping iteration at j = 2
# Inside the loop at j = 3
# Inside the loop at j = 4

# pass example
for k in range(3):
    if k == 1:
        print("Doing nothing at k =", k)
        pass
    else:
        print("Inside the loop at k =", k)

# Output:
# Inside the loop at k = 0
# Doing nothing at k = 1
# Inside the loop at k = 2
```

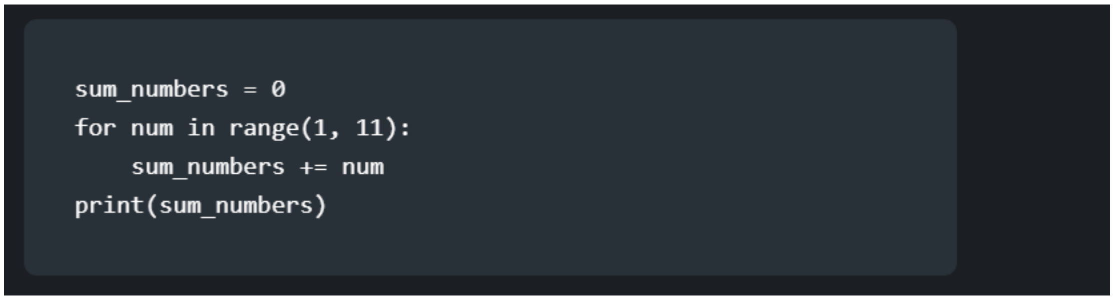
Section-C

1. **Print numbers from 1 to 10 using a for loop**:



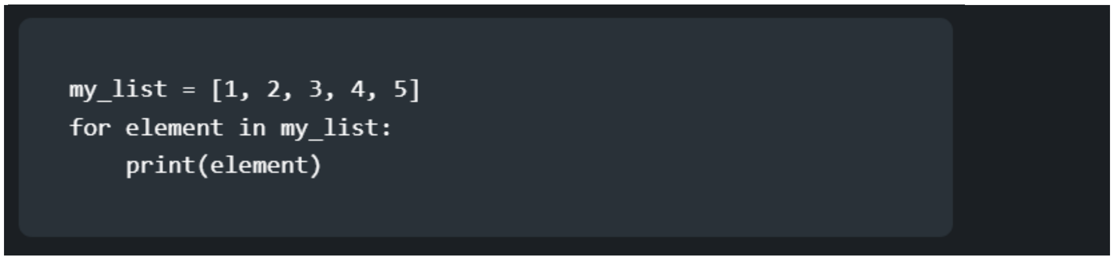
```python
for num in range(1, 11):
    print(num)
```

2. **Calculate the sum of numbers from 1 to 10 using a for loop**:



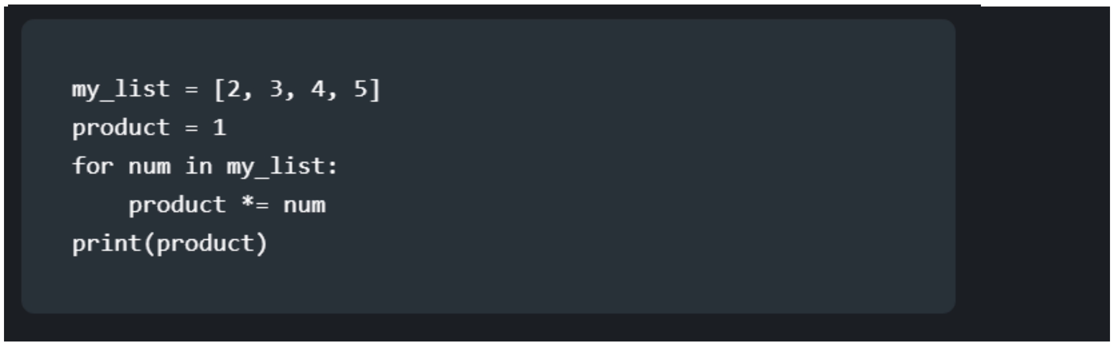
```python
sum_numbers = 0
for num in range(1, 11):
    sum_numbers += num
print(sum_numbers)
```

3. **Print the elements of a list using a for loop**:



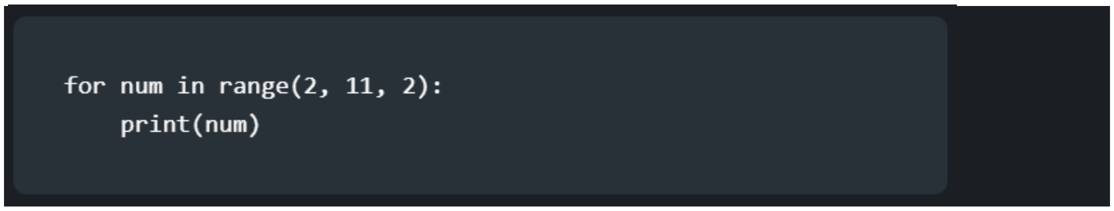
```python
my_list = [1, 2, 3, 4, 5]
for element in my_list:
    print(element)
```

4. **Calculate the product of elements in a list using a for loop**:



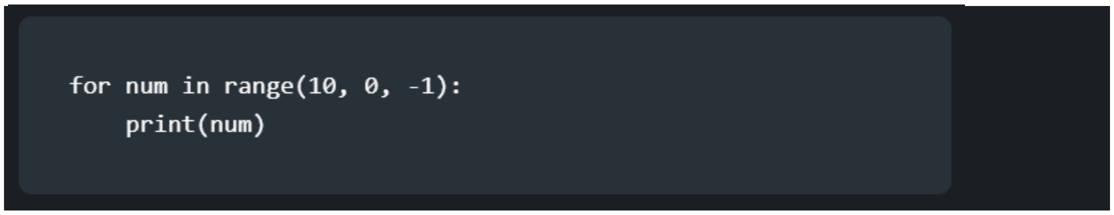
```python
my_list = [2, 3, 4, 5]
product = 1
for num in my_list:
    product *= num
print(product)
```

5. **Print even numbers from 1 to 10 using a for loop**:



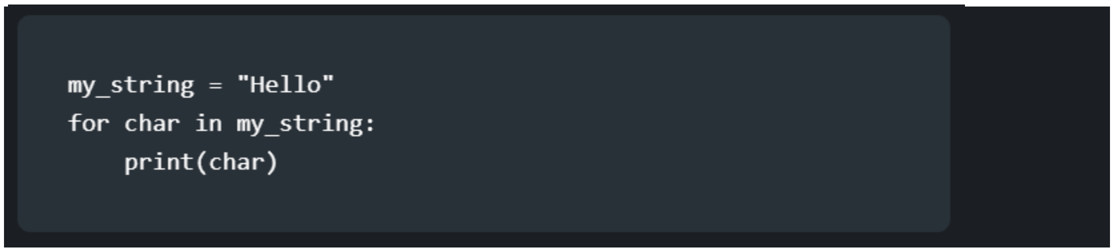
```python
for num in range(2, 11, 2):
    print(num)
```

6. **Print numbers in reverse from 10 to 1 using a for loop**:



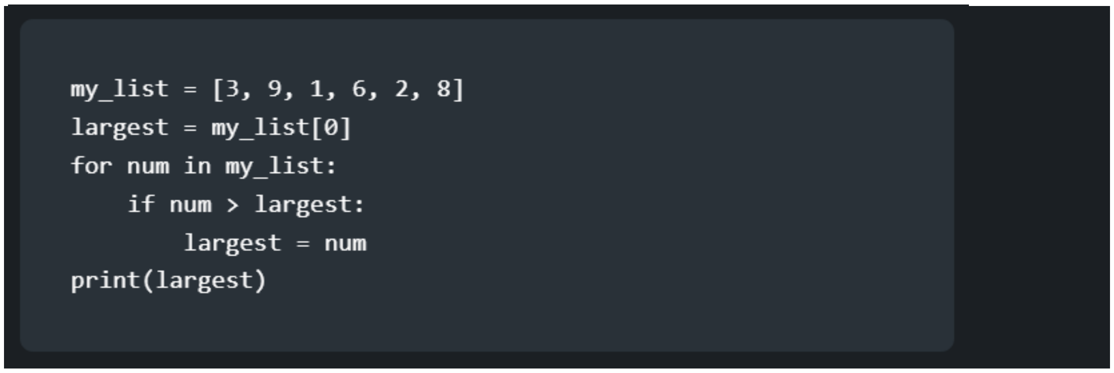
```python
for num in range(10, 0, -1):
    print(num)
```

7. **Print characters of a string using a for loop**:



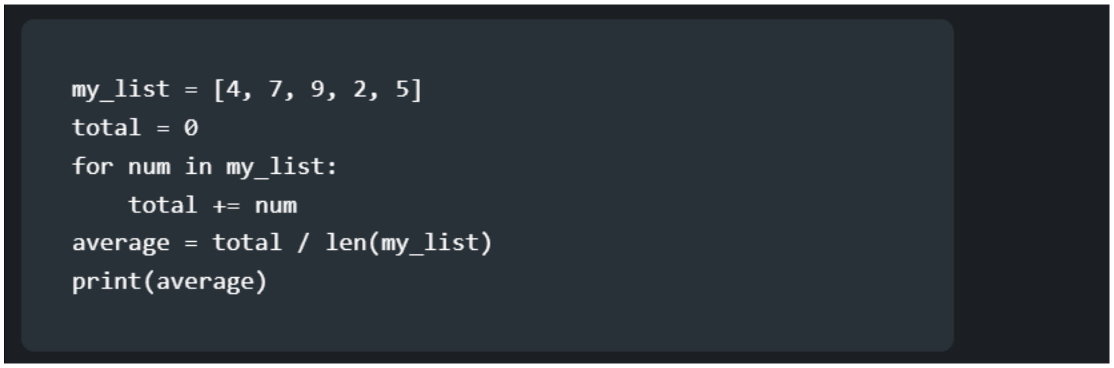
```python
my_string = "Hello"
for char in my_string:
    print(char)
```

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



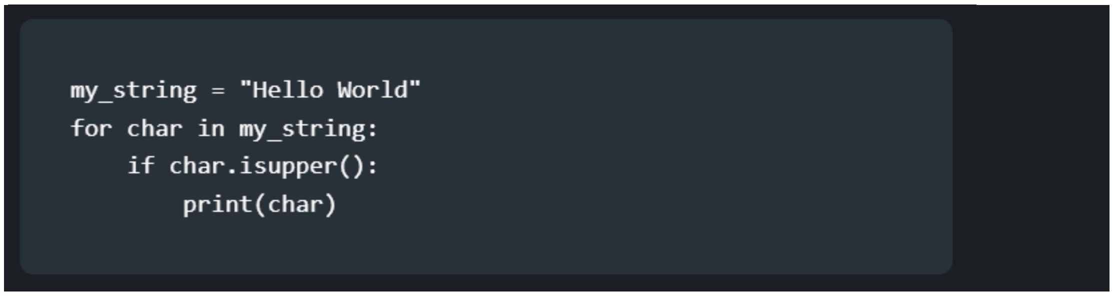
```python
my_list = [3, 9, 1, 6, 2, 8]
largest = my_list[0]
for num in my_list:
    if num > largest:
        largest = num
print(largest)
```

9. **Find the average of numbers in a list using a for loop**:



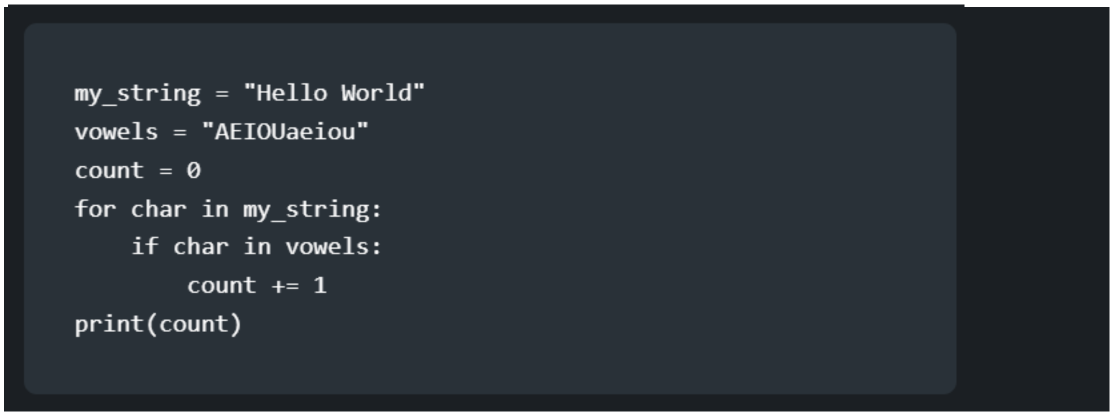
```python
my_list = [4, 7, 9, 2, 5]
total = 0
for num in my_list:
    total += num
average = total / len(my_list)
print(average)
```

10. **Print all uppercase letters in a string using a for loop**:



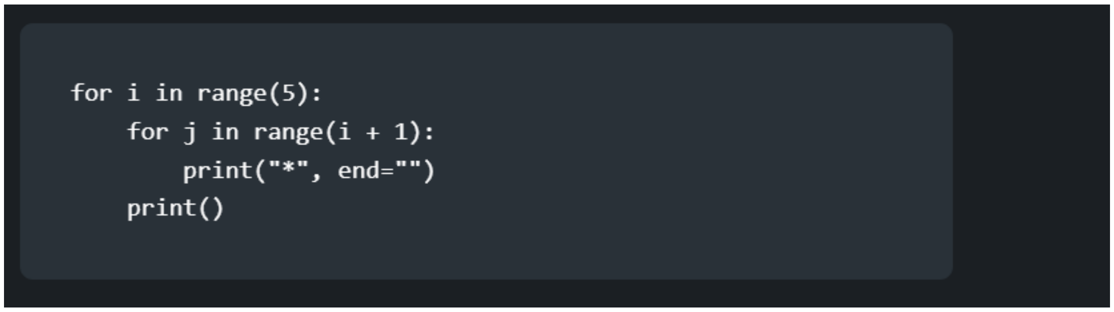
```python
my_string = "Hello World"
for char in my_string:
    if char.isupper():
        print(char)
```

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



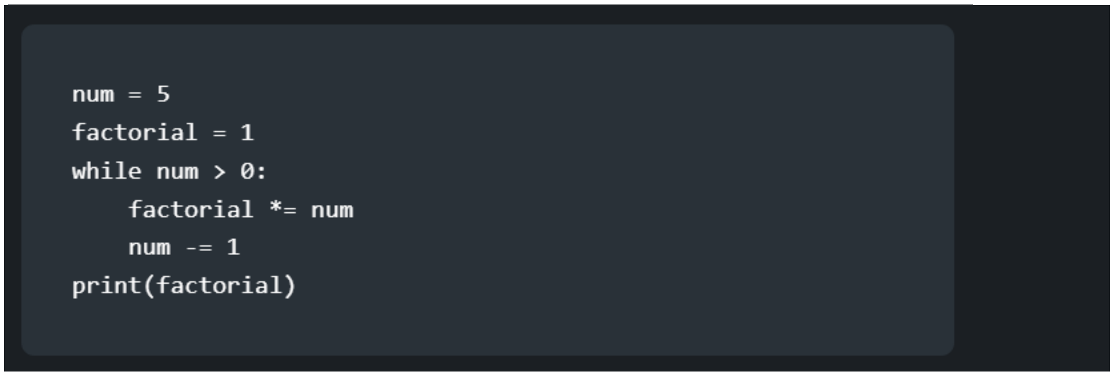
```python
my_string = "Hello World"
vowels = "AEIOUaeiou"
count = 0
for char in my_string:
    if char in vowels:
        count += 1
print(count)
```

12. **Print a pattern of stars using nested for loops**:



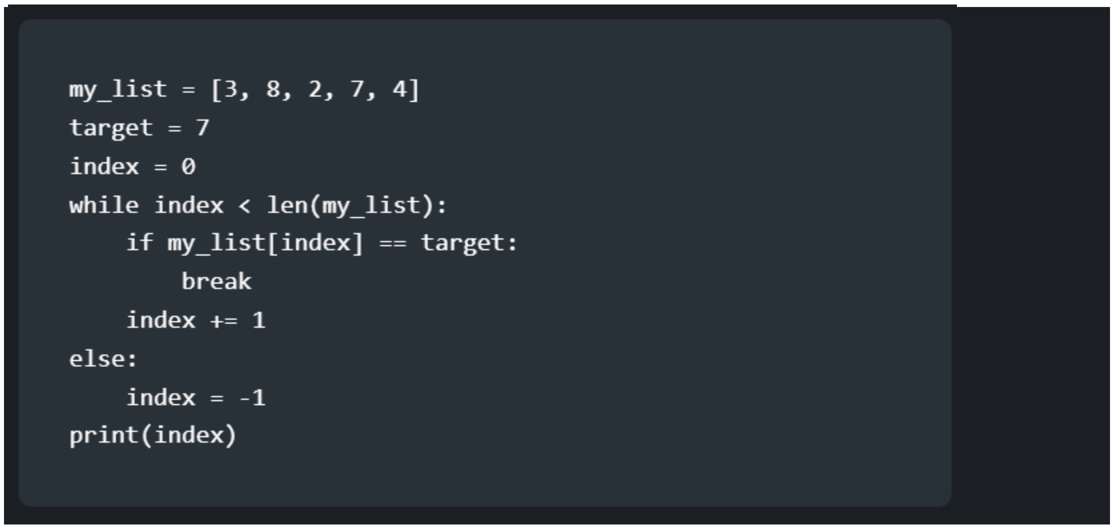
```
for i in range(5):
    for j in range(i + 1):
        print("*", end="")
    print()
```

13. **Calculate factorial of a number using a while loop**:



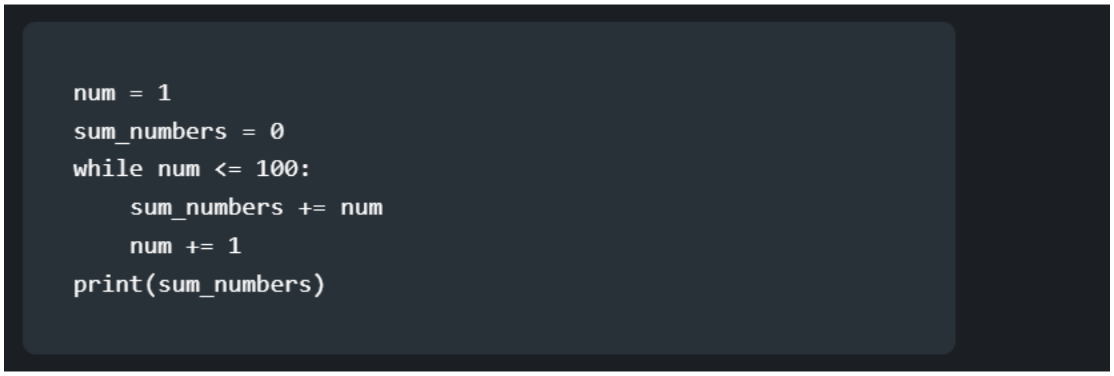
```
num = 5
factorial = 1
while num > 0:
    factorial *= num
    num -= 1
print(factorial)
```

14. **Find the first occurrence of a number in a list using a while loop**:



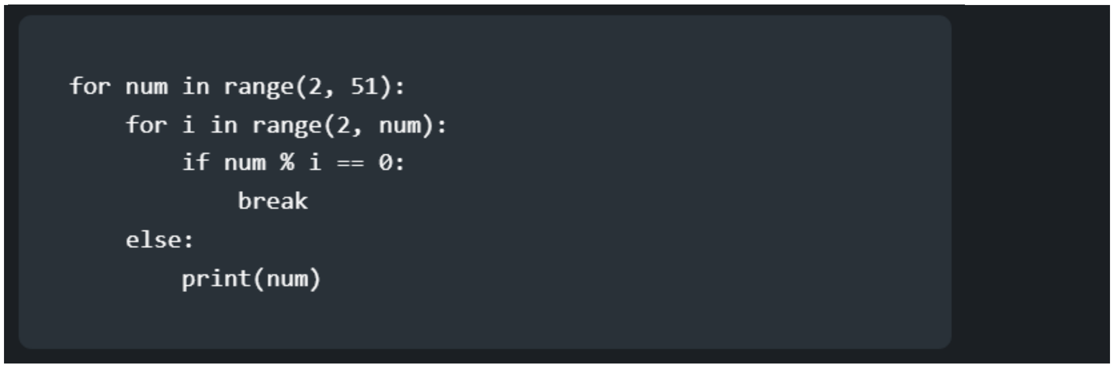
```
my_list = [3, 8, 2, 7, 4]
target = 7
index = 0
while index < len(my_list):
    if my_list[index] == target:
        break
    index += 1
else:
    index = -1
print(index)
```

15. **Calculate the sum of numbers from 1 to 100 using a while loop**:



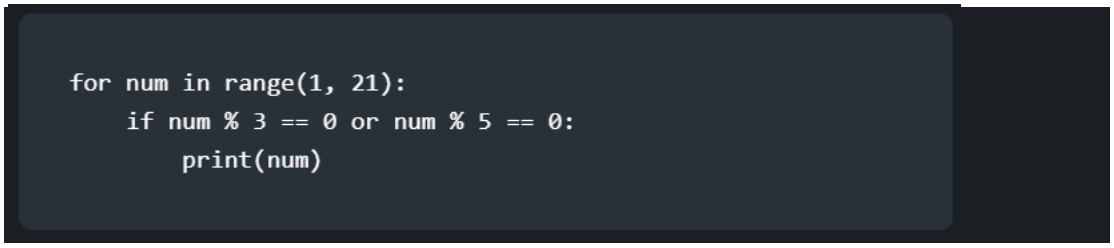
```python
num = 1
sum_numbers = 0
while num <= 100:
    sum_numbers += num
    num += 1
print(sum_numbers)
```

16. **Find all prime numbers between 1 and 50 using nested for and if**:



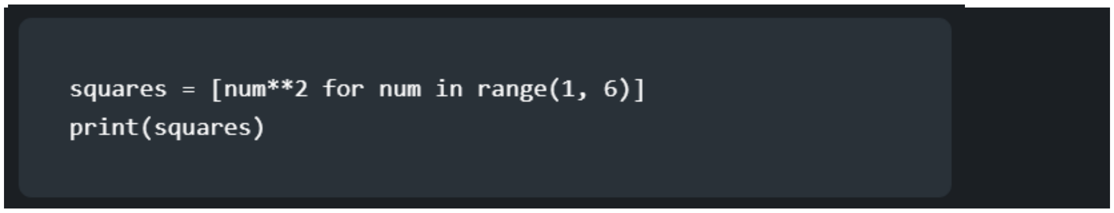
```python
for num in range(2, 51):
    for i in range(2, num):
        if num % i == 0:
            break
    else:
        print(num)
```

17. **Print numbers divisible by 3 or 5 from 1 to 20 using a for loop**:



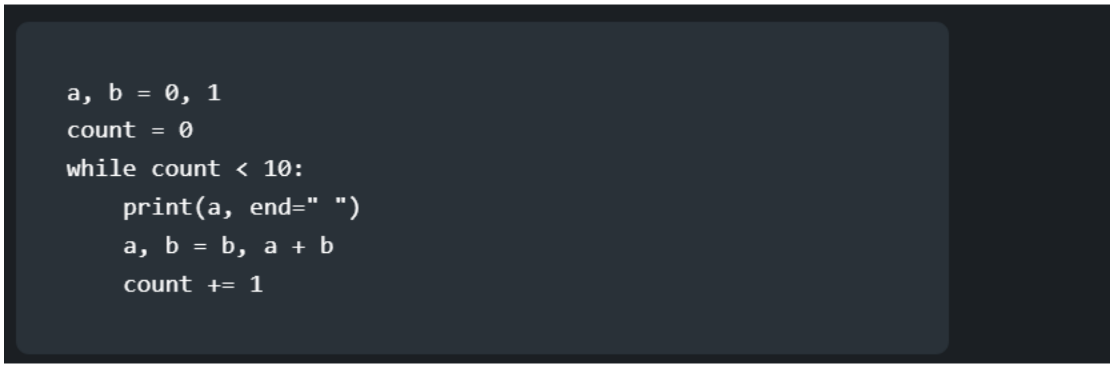
```python
for num in range(1, 21):
    if num % 3 == 0 or num % 5 == 0:
        print(num)
```

18. **Print a list of squares of numbers from 1 to 5 using a list comprehension**:



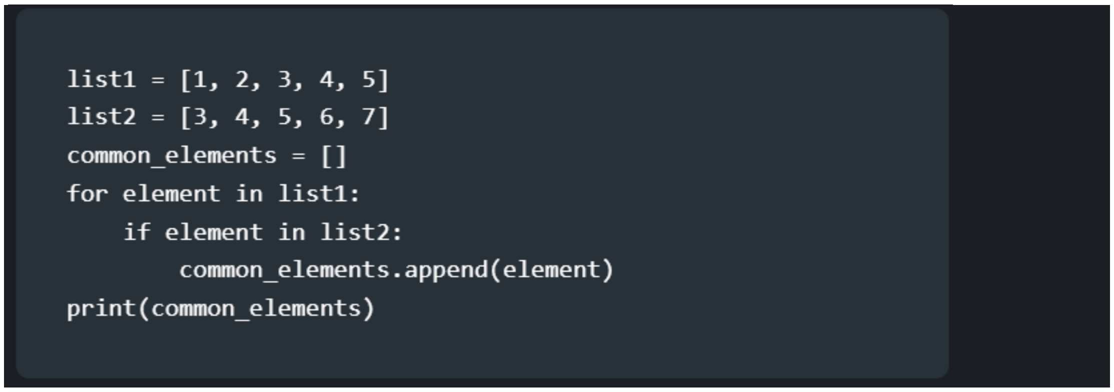
```python
squares = [num**2 for num in range(1, 6)]
print(squares)
```

19. **Print the Fibonacci sequence up to the 10th term using a while loop**:



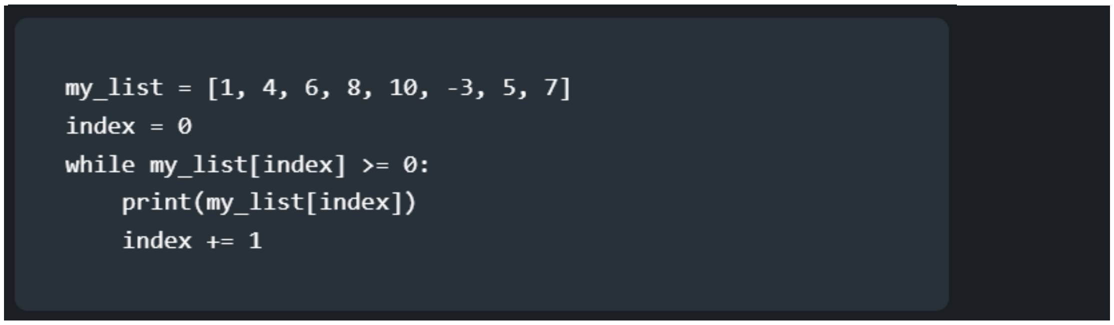
```python
a, b = 0, 1
count = 0
while count < 10:
    print(a, end=" ")
    a, b = b, a + b
    count += 1
```

20. **Find the common elements in two lists using a for loop**:



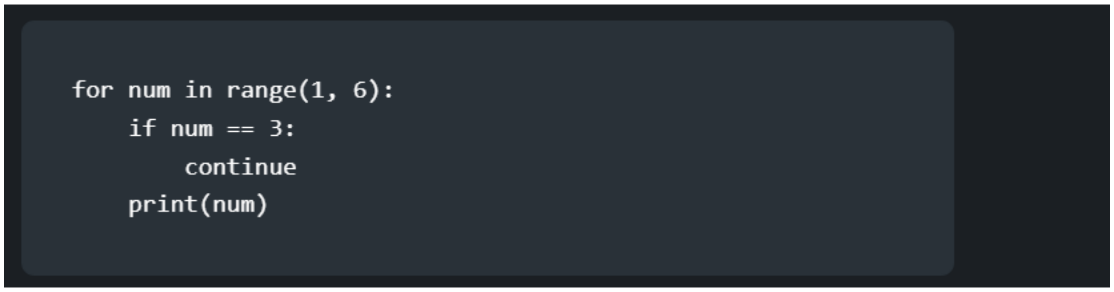
```python
list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]
common_elements = []
for element in list1:
    if element in list2:
        common_elements.append(element)
print(common_elements)
```

21. **Print numbers in a list until a negative number is encountered using a while loop**:



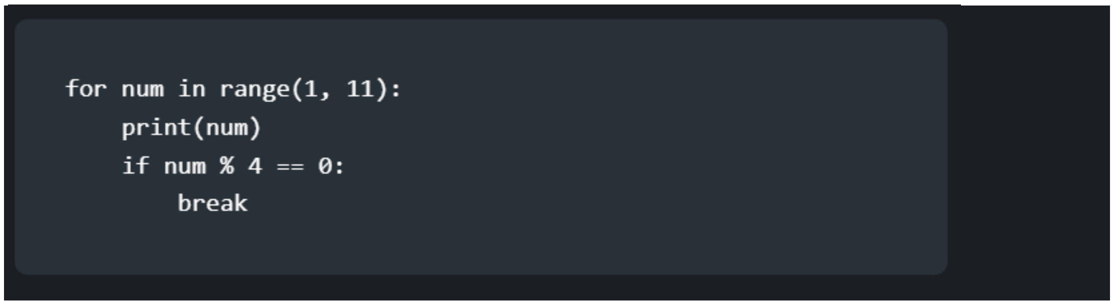
```python
my_list = [1, 4, 6, 8, 10, -3, 5, 7]
index = 0
while my_list[index] >= 0:
    print(my_list[index])
    index += 1
```

22. **Print numbers from 1 to 5, except 3 using a for loop and continue statement**:



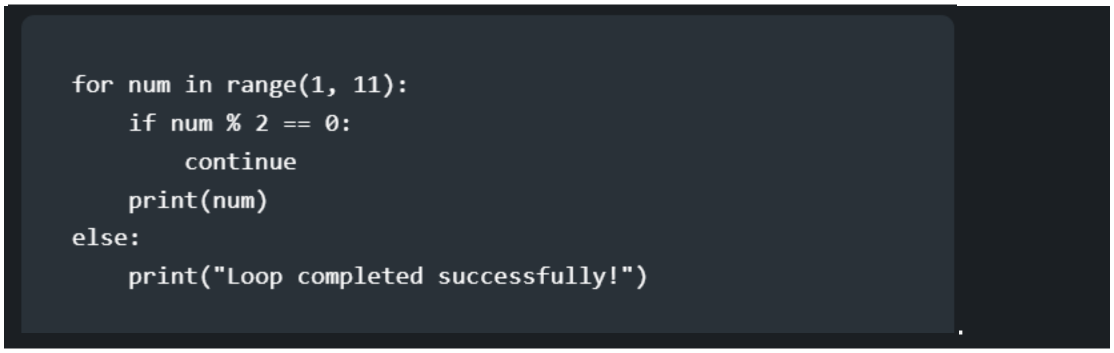
```python
for num in range(1, 6):
    if num == 3:
        continue
    print(num)
```

23. **Print numbers from 1 to 10. If a number is divisible by 4, stop the loop using a for loop and break statement**:



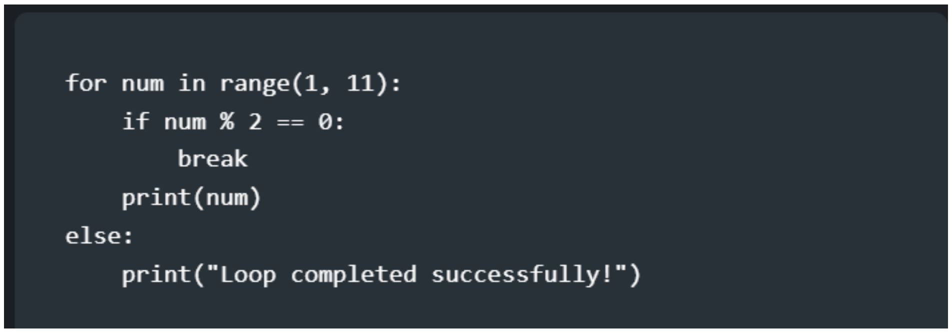
```python
for num in range(1, 11):
    print(num)
    if num % 4 == 0:
        break
```

24. **Print numbers from 1 to 10. If a number is even, skip it using a for loop and else clause**:



```python
for num in range(1, 11):
    if num % 2 == 0:
        continue
    print(num)
else:
    print("Loop completed successfully!")
```

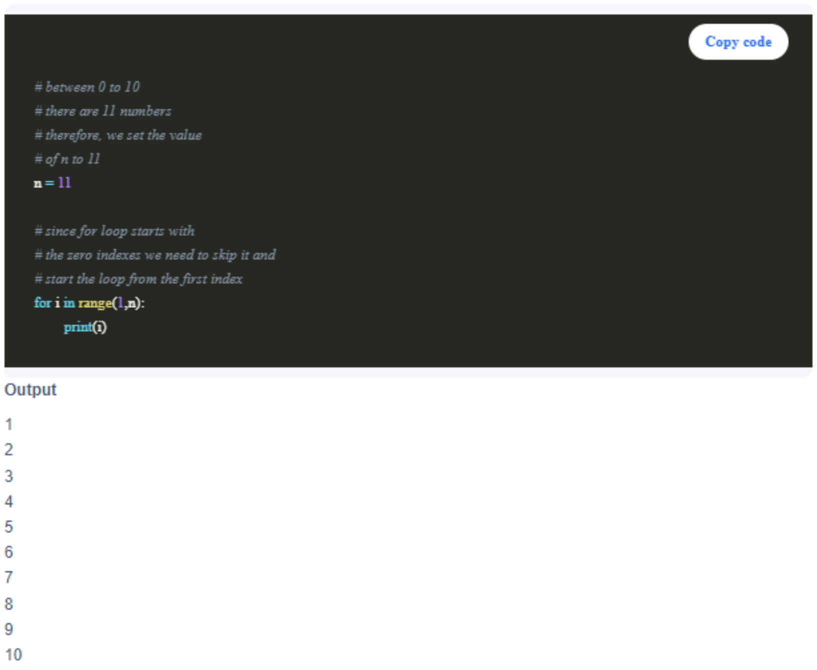
25. **Print numbers from 1 to 10. If a number is even, break the loop using a for loop and else clause**:



```python
for num in range(1, 11):
    if num % 2 == 0:
        break
    print(num)
else:
    print("Loop completed successfully!")
```

Section - D

1.

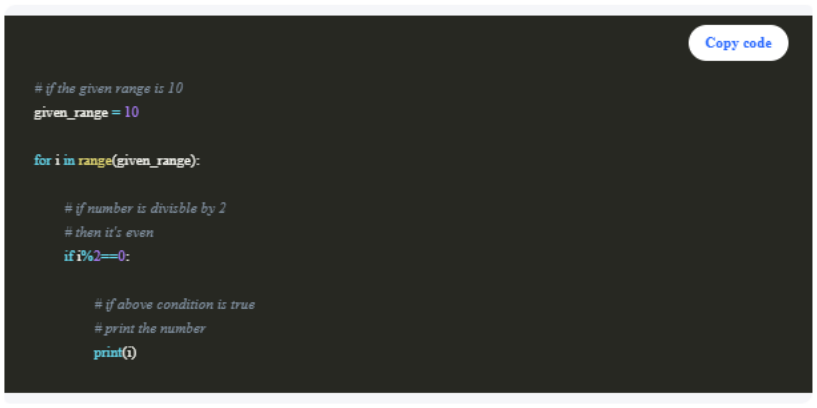


```python
# between 0 to 10
# there are 11 numbers
# therefore, we set the value
# of n to 11
n = 11

# since for loop starts with
# the zero indexes we need to skip it and
# start the loop from the first index
for i in range(1,n):
    print(i)
```

Output

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

2.

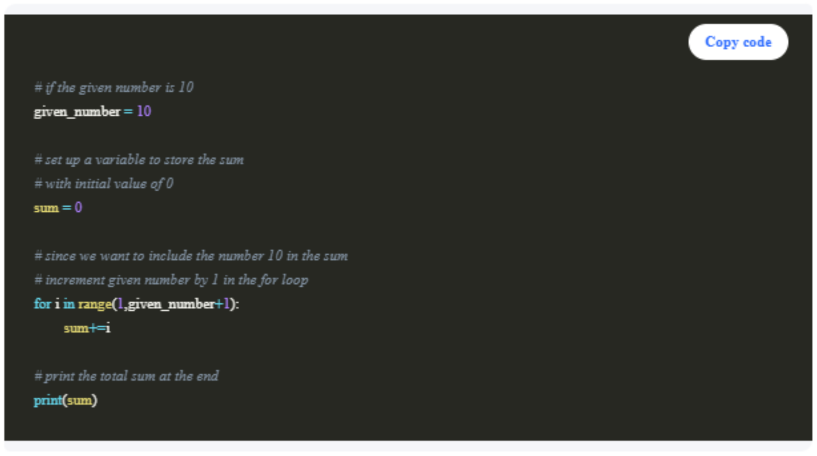


```python
# if the given range is 10
given_range = 10

for i in range(given_range):

    # if number is divisble by 2
    # then it's even
    if i%2==0:

        # if above condition is true
        # print the number
        print(i)
```

Output

```
0
2
4
6
```

3.

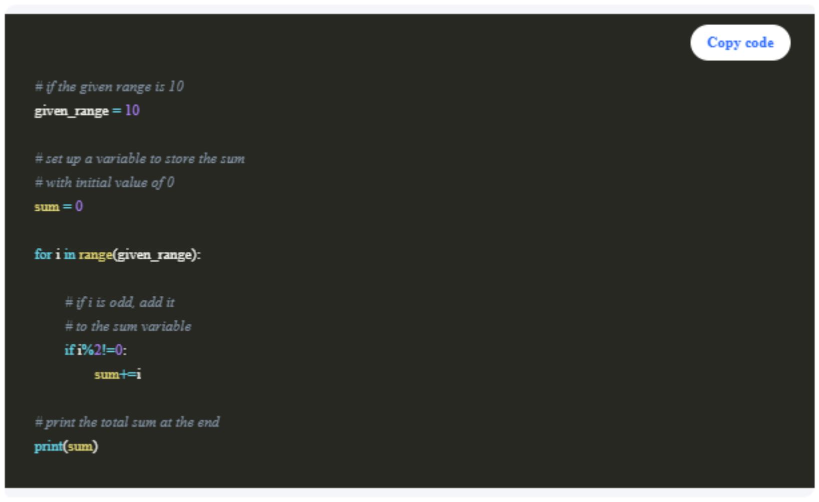


```python
# if the given number is 10
given_number = 10

# set up a variable to store the sum
# with initial value of 0
sum = 0

# since we want to include the number 10 in the sum
# increment given number by 1 in the for loop
for i in range(1,given_number+1):
    sum+=i

# print the total sum at the end
print(sum)
```

Output

```
55
```

4.

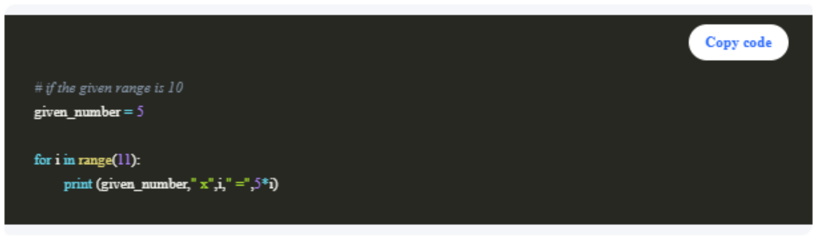


```
# if the given range is 10
given_range = 10

# set up a variable to store the sum
# with initial value of 0
sum = 0

for i in range(given_range):

    # if i is odd, add it
    # to the sum variable
    if i%2!=0:
        sum+=i

# print the total sum at the end
print(sum)
```

Output

```
25
```

5.

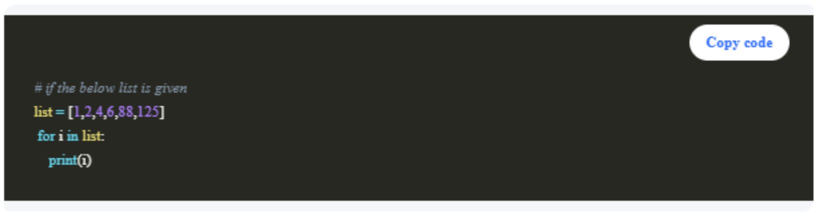


```
# if the given range is 10
given_number = 5

for i in range(11):
    print (given_number," x",i," =",5*i)
```

Output

```
5 x 0 = 0
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

6.

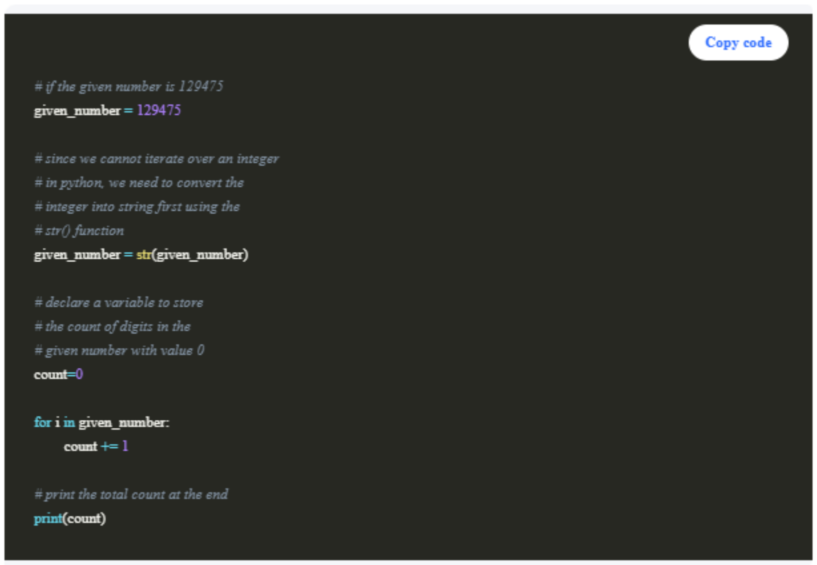


```
# if the below list is given
list = [1,2,4,6,88,125]
 for i in list:
   print(i)
```

Output

```
1
2
4
6
88
125
```

7.

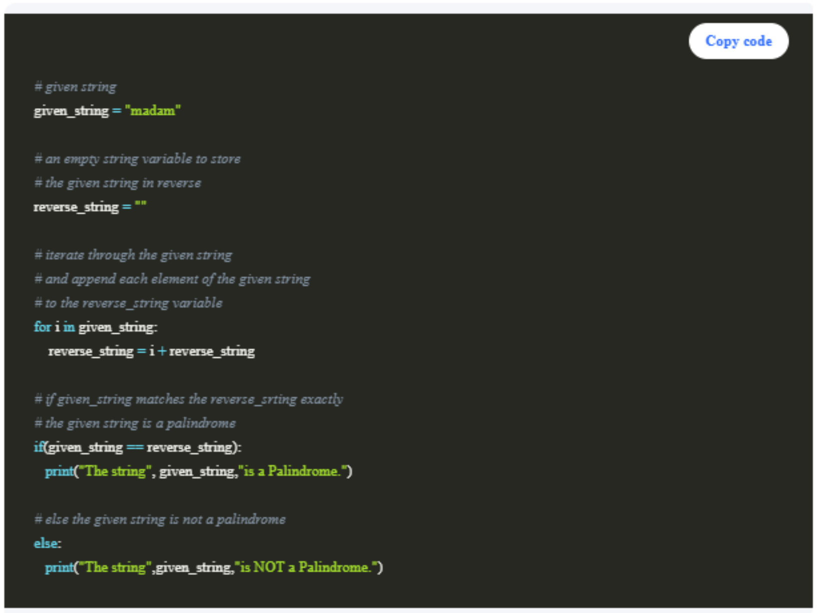


```
# if the given number is 129475
given_number = 129475

# since we cannot iterate over an integer
# in python, we need to convert the
# integer into string first using the
# str() function
given_number = str(given_number)

# declare a variable to store
# the count of digits in the
# given number with value 0
count=0

for i in given_number:
      count += 1

# print the total count at the end
print(count)
```

Output

```
6
```

8.

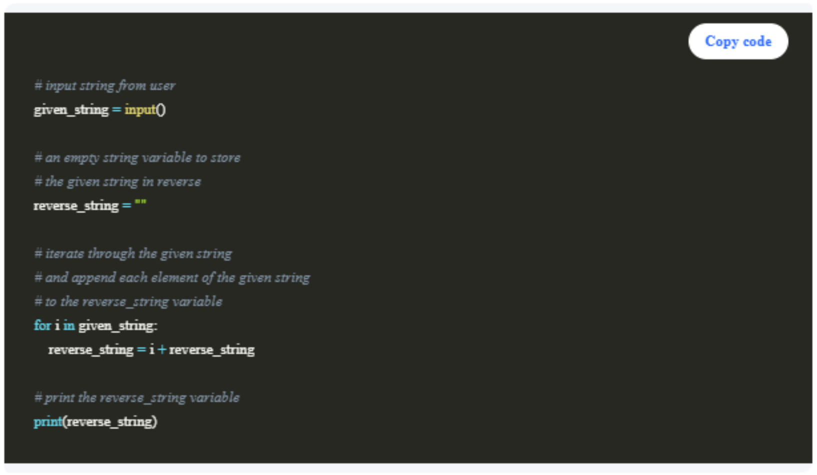


```
# given string
given_string = "madam"

# an empty string variable to store
# the given string in reverse
reverse_string = ""

# iterate through the given string
# and append each element of the given string
# to the reverse_string variable
for i in given_string:
   reverse_string = i + reverse_string

# if given_string matches the reverse_srting exactly
# the given string is a palindrome
if(given_string == reverse_string):
  print("The string", given_string,"is a Palindrome.")

# else the given string is not a palindrome
else:
  print("The string",given_string,"is NOT a Palindrome.")
```

Output

```
The string madam is a Palindrome String.
```

9.



```
# input string from user
given_string = input()

# an empty string variable to store
# the given string in reverse
reverse_string = ""

# iterate through the given string
# and append each element of the given string
# to the reverse_string variable
for i in given_string:
    reverse_string = i + reverse_string

# print the reverse_string variable
print(reverse_string)
```

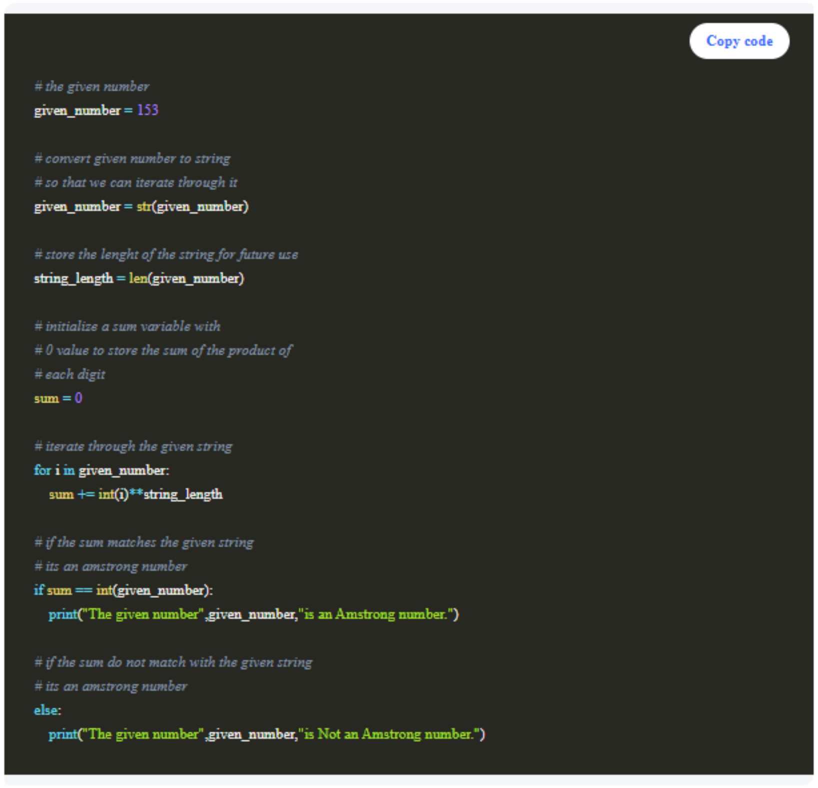
Input

Naukri

Output

irkuaN

10.

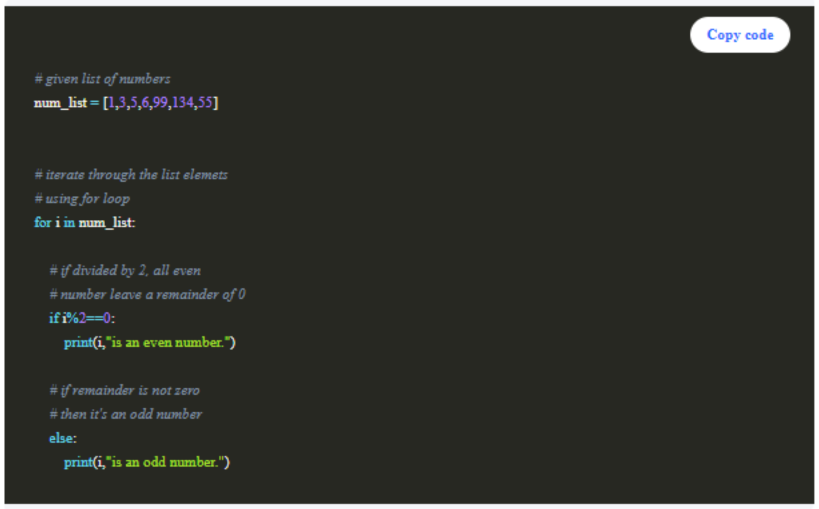


```
# the given number
given_number = 153

# convert given number to string
# so that we can iterate through it
given_number = str(given_number)

# store the lenght of the string for future use
string_length = len(given_number)

# initialize a sum variable with
# 0 value to store the sum of the product of
# each digit
sum = 0

# iterate through the given string
for i in given_number:
    sum += int(i)**string_length

# if the sum matches the given string
# its an amstrong number
if sum == int(given_number):
    print("The given number",given_number,"is an Amstrong number.")

# if the sum do not match with the given string
# its an amstrong number
else:
    print("The given number",given_number,"is Not an Amstrong number.")
```

Output

The given number 153 is an Amstrong number.

11.

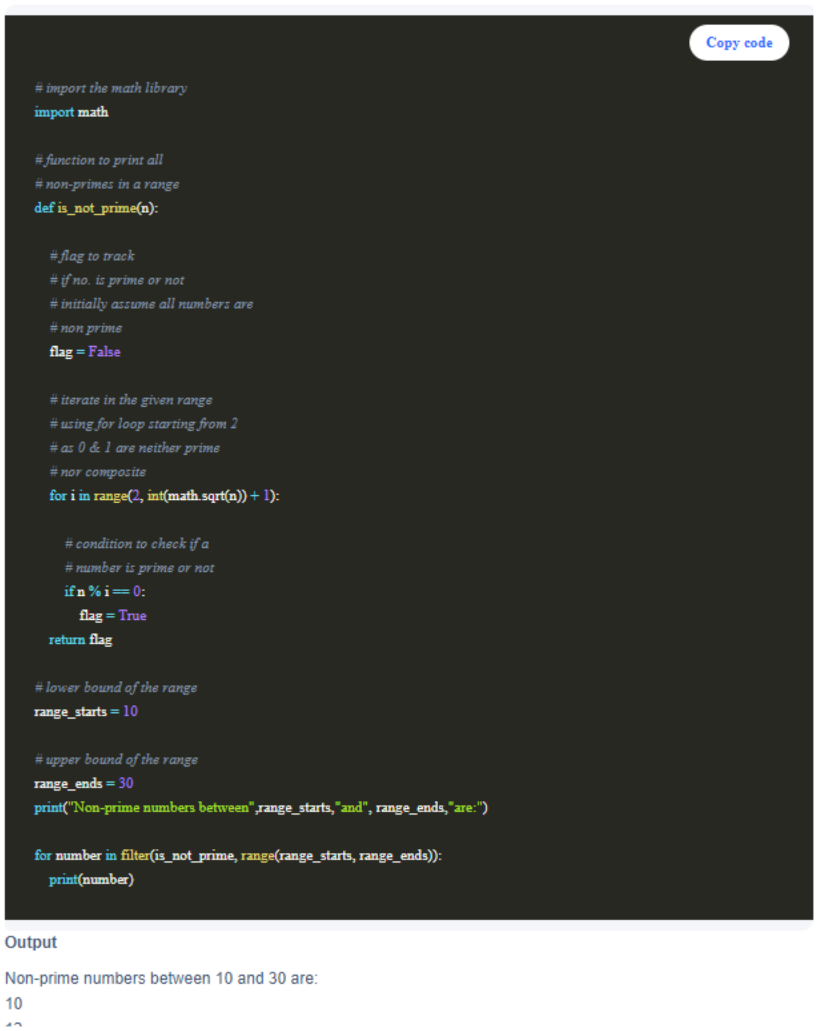


```
# given list of numbers
num_list = [1,3,5,6,99,134,55]


# iterate through the list elemets
# using for loop
for i in num_list:

    # if divided by 2, all even
    # number leave a remainder of 0
    if i%2==0:
        print(i,"is an even number.")

    # if remainder is not zero
    # then it's an odd number
    else:
        print(i,"is an odd number.")
```

Output

```
1 is an odd number.
3 is an odd number.
5 is an odd number.
6 is an even number.
99 is an odd number.
134 is an even number.
55 is an odd number.
```

12.

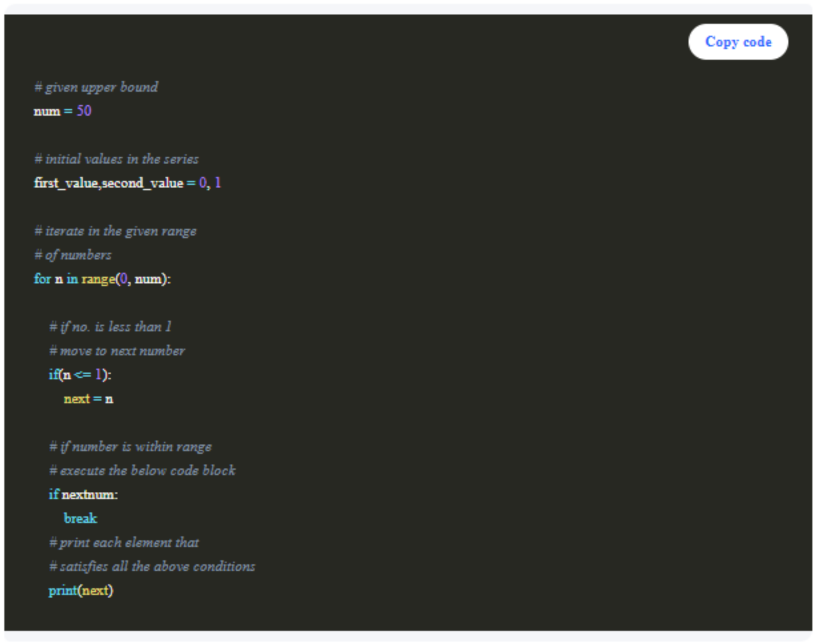


```python
# import the math library
import math

# function to print all
# non-primes in a range
def is_not_prime(n):

    # flag to track
    # if no. is prime or not
    # initially assume all numbers are
    # non prime
    flag = False

    # iterate in the given range
    # using for loop starting from 2
    # as 0 & 1 are neither prime
    # nor composite
    for i in range(2, int(math.sqrt(n)) + 1):

        # condition to check if a
        # number is prime or not
        if n % i == 0:
            flag = True
    return flag

# lower bound of the range
range_starts = 10

# upper bound of the range
range_ends = 30
print("Non-prime numbers between",range_starts,"and", range_ends,"are:")

for number in filter(is_not_prime, range(range_starts, range_ends)):
    print(number)
```

Output

```
Non-prime numbers between 10 and 30 are:
10
```

13.

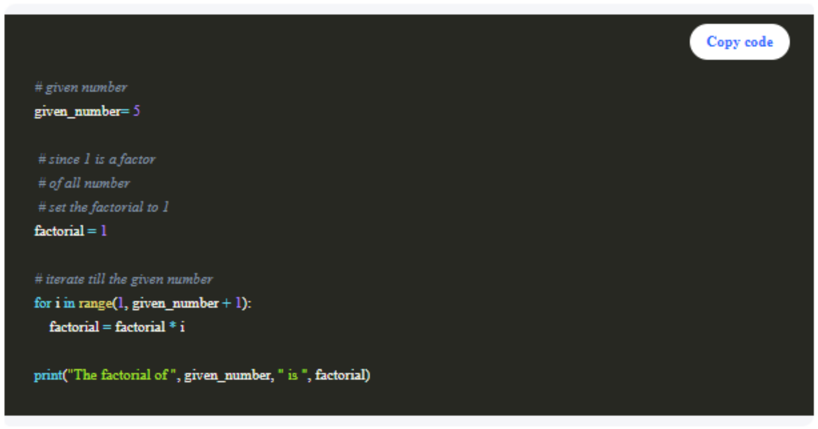


```
# given upper bound
num = 50

# initial values in the series
first_value,second_value = 0, 1

# iterate in the given range
# of numbers
for n in range(0, num):

  # if no. is less than 1
  # move to next number
  if(n <= 1):
    next = n

  # if number is within range
  # execute the below code block
  if nextnum:
    break
  # print each element that
  # satisfies all the above conditions
  print(next)
```

Output

```
1
2
3
5
8
13
21
34
```

14.

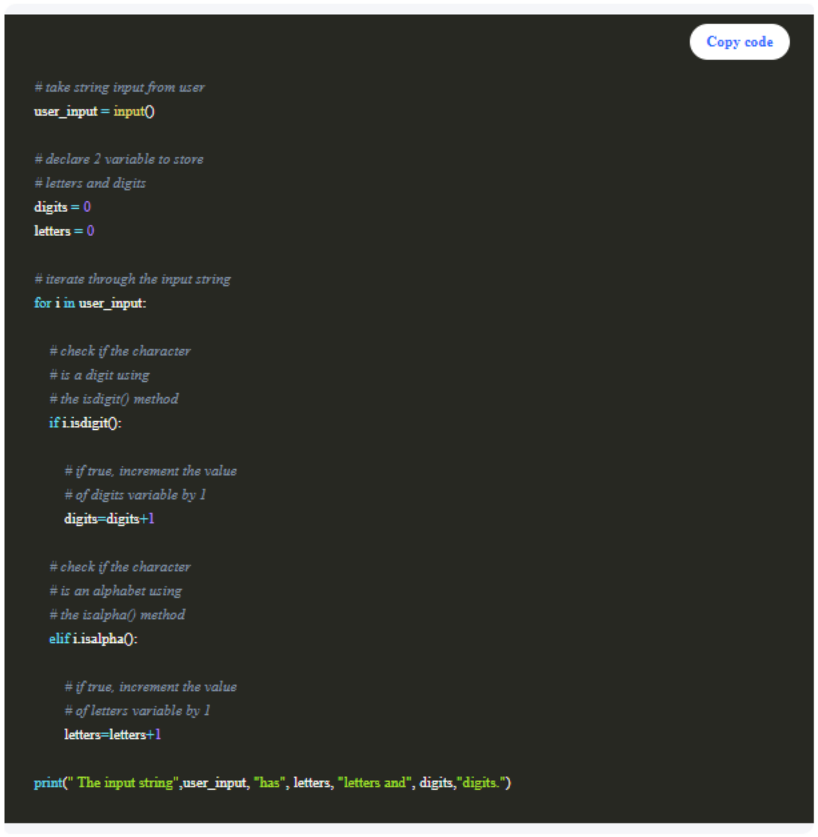


```
# given number
given_number= 5

# since 1 is a factor
# of all number
# set the factorial to 1
factorial = 1

# iterate till the given number
for i in range(1, given_number + 1):
  factorial = factorial * i

print("The factorial of ", given_number, " is ", factorial)
```

Output

```
The factorial of 5 is 120
```

15.



```python
# take string input from user
user_input = input()

# declare 2 variable to store
# letters and digits
digits = 0
letters = 0

# iterate through the input string
for i in user_input:

    # check if the character
    # is a digit using
    # the isdigit() method
    if i.isdigit():

        # if true, increment the value
        # of digits variable by 1
        digits=digits+1

    # check if the character
    # is an alphabet using
    # the isalpha() method
    elif i.isalpha():

        # if true, increment the value
        # of letters variable by 1
        letters=letters+1

print(" The input string",user_input, "has", letters, "letters and", digits,"digits.")
```

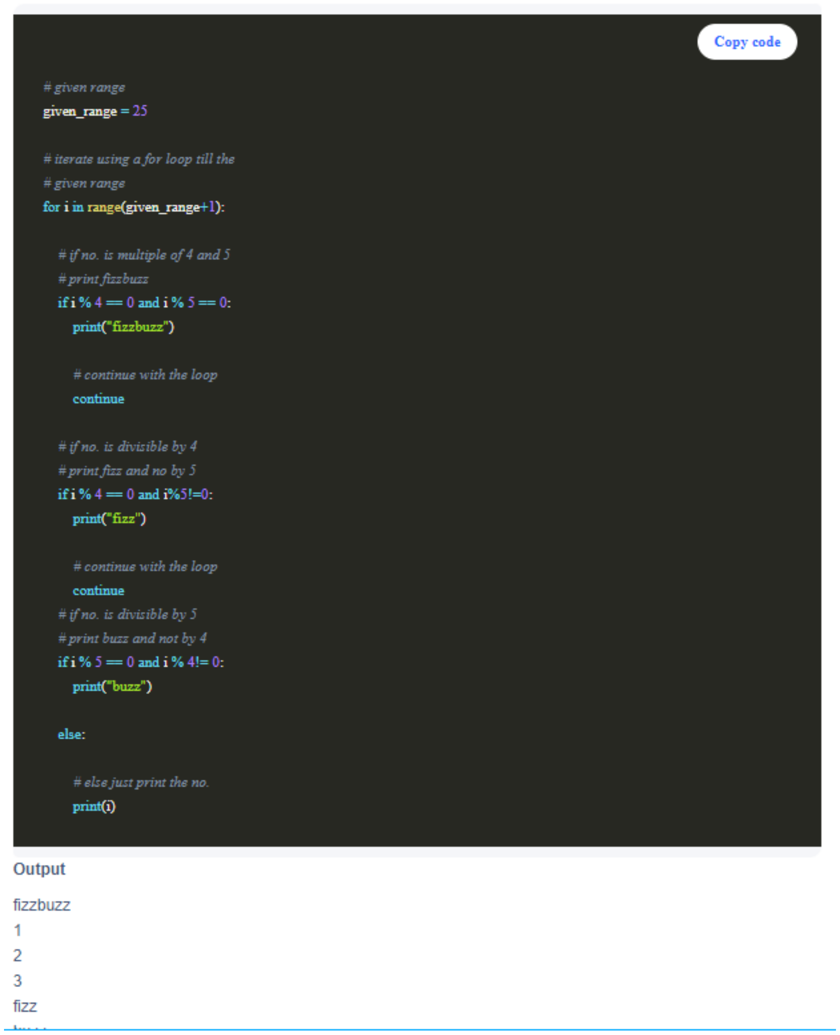
Input

Naukri1234

Output

The input string Naukri12345 has 6 letters and 5 digits.

16.

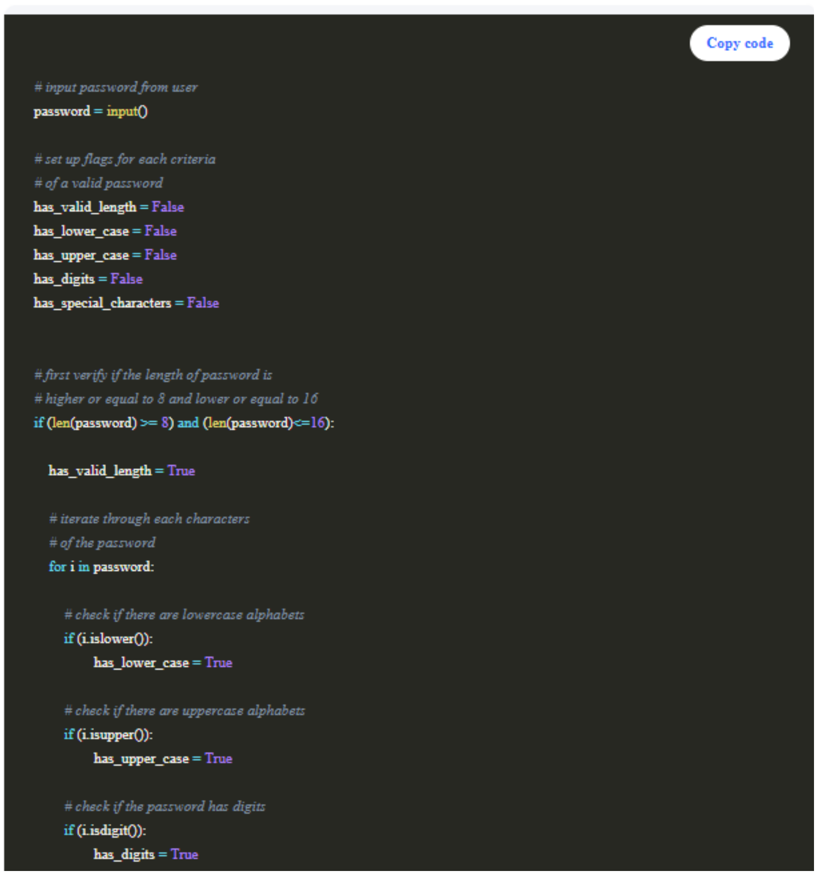


```python
# given range
given_range = 25

# iterate using a for loop till the
# given range
for i in range(given_range+1):

    # if no. is multiple of 4 and 5
    # print fizzbuzz
    if i % 4 == 0 and i % 5 == 0:
        print("fizzbuzz")

        # continue with the loop
        continue

    # if no. is divisible by 4
    # print fizz and no by 5
    if i % 4 == 0 and i%5!=0:
        print("fizz")

        # continue with the loop
        continue
    # if no. is divisible by 5
    # print buzz and not by 4
    if i % 5 == 0 and i % 4!= 0:
        print("buzz")

    else:

        # else just print the no.
        print(i)
```

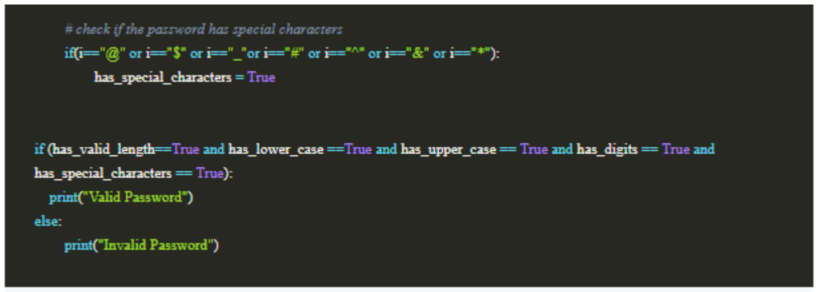
Output

```
fizzbuzz
1
2
3
fizz
```

17.



```
# input password from user
password = input()

# set up flags for each criteria
# of a valid password
has_valid_length = False
has_lower_case = False
has_upper_case = False
has_digits = False
has_special_characters = False


# first verify if the length of password is
# higher or equal to 8 and lower or equal to 16
if (len(password) >= 8) and (len(password)<=16):

    has_valid_length = True

    # iterate through each characters
    # of the password
    for i in password:

        # check if there are lowercase alphabets
        if (i.islower()):
            has_lower_case = True

        # check if there are uppercase alphabets
        if (i.isupper()):
            has_upper_case = True

        # check if the password has digits
        if (i.isdigit()):
            has_digits = True
```



```
        # check if the password has special characters
        if(i=="@" or i=="$" or i=="_"or i=="#" or i=="^" or i=="&" or i=="*"):
            has_special_characters = True


if (has_valid_length==True and has_lower_case ==True and has_upper_case == True and has_digits == True and
has_special_characters == True):
    print("Valid Password")
else:
        print("Invalid Password")
```

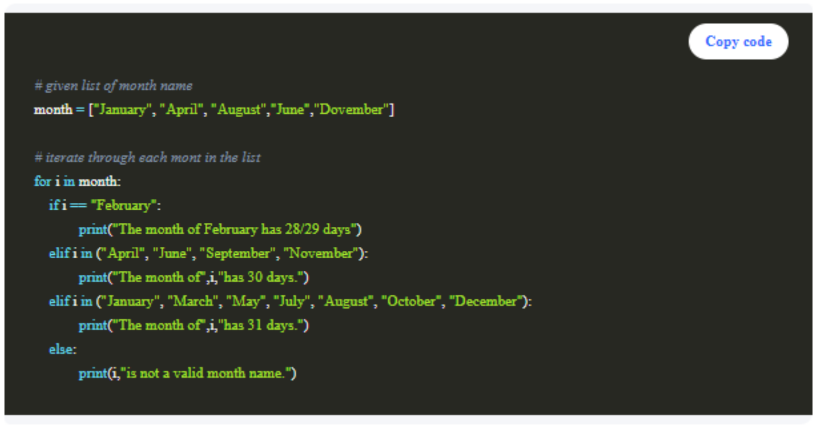
Input

Naukri12345@

Output

Naukri12345@

18.



```
# given list of month name
month = ["January", "April", "August","June","Dovember"]

# iterate through each mont in the list
for i in month:
  if i == "February":
        print("The month of February has 28/29 days")
  elif i in ("April", "June", "September", "November"):
        print("The month of",i,"has 30 days.")
  elif i in ("January", "March", "May", "July", "August", "October", "December"):
        print("The month of",i,"has 31 days.")
  else:
        print(i,"is not a valid month name.")
```

Output

```
The month of January has 31 days.
The month of April has 30 days.
The month of August has 31 days.
The month of June has 30 days.
November is not a valid month name
```