1. Assign the value 7 to the variable guess\_me. Then, write the conditional tests (if, else, and elif) to print the string 'too low' if guess\_me is less than 7, 'too high' if greater than 7, and 'just right' if equal to 7.

Ans : Certainly! Here's an example of how you can assign the value 7 to the variable guess\_me and write the conditional tests to print the corresponding strings based on its value:

pythonCopy code

guess\_me = 7 if guess\_me < 7: print('too low') elif guess\_me > 7: print('too high') else: print('just right')

Output:

cssCopy code

just right

In this example, the value 7 is assigned to the variable **guess\_me**. The conditional tests are then written using **if**, **elif** (short for "else if"), and **else** statements. If **guess\_me** is less than 7, the string **'too low'** is printed. If it is greater than 7, the string **'too high'** is printed. If it is equal to 7, the string **'just right'** is printed. Since the value of **guess\_me** is 7 in this example, the output is **'just right'**.

2. Assign the value 7 to the variable guess\_me and the value 1 to the variable start. Write a while loop that compares start with guess\_me. Print too low if start is less than guess me. If start equals guess\_me, print 'found it!' and exit the loop. If start is greater than guess\_me, print 'oops' and exit the loop. Increment start at the end of the loop.

Ans : Certainly! Here's an example of how you can assign the values 7 and 1 to the variables guess\_me and start, respectively. Then, you can write a while loop to compare start with guess\_me and perform the desired actions based on the conditions:

guess\_me = 7 start = 1 while start <= guess\_me: if start < guess\_me: print('too low') elif start == guess\_me: print('found it!') break else: print('oops') break start += 1

Output:

too low too low too low too low too low too low found it!

In this example, the while loop compares the value of **start** with **guess\_me**. If **start** is less than **guess\_me**, it prints **'too low'**. If **start** is equal to **guess\_me**, it prints **'found it!'** and exits the loop using the **break** statement. If **start** is greater than **guess\_me**, it prints **'oops'** and also exits the loop. After each iteration, **start** is incremented by 1 using **start += 1**.

3. Print the following values of the list [3, 2, 1, 0] using a for loop.

Ans : Certainly! Here's an example of how you can use a for loop to print the values of the list [3, 2, 1, 0]:

pythonCopy code

my\_list = [3, 2, 1, 0] for value in my\_list: print(value)

Output:

Copy code

3 2 1 0

In this example, the for loop iterates over each value in the **my\_list** list. The variable **value** takes on each value in the list sequentially, and it is printed within the loop. This results in printing each value of the list **[3, 2, 1, 0]** on a separate line.

4. Use a list comprehension to make a list of the even numbers in range(10)

Ans :Certainly! Here's an example of how you can use a list comprehension to create a list of the even numbers in the range from 0 to 9 (range(10)):

even\_numbers = [num for num in range(10) if num % 2 == 0] print(even\_numbers)

Output:

[0, 2, 4, 6, 8]

In this example, the list comprehension **[num for num in range(10) if num % 2 == 0]** creates a new list containing only the even numbers from the range of 0 to 9. The condition **if num % 2 == 0** checks if the number is divisible by 2 without a remainder, indicating it's an even number. The resulting list **[0, 2, 4, 6, 8]** is then printed.

5. Use a dictionary comprehension to create the dictionary squares. Use range(10) to return the keys, and use the square of each key as its value.

Ans : Certainly! Here's an example of how you can use a dictionary comprehension to create the squares dictionary where the keys are taken from the range 0 to 9 (range(10)), and the values are the squares of each key:

squares = {num: num \*\* 2 for num in range(10)} print(squares)

Output:

{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

In this example, the dictionary comprehension **{num: num \*\* 2 for num in range(10)}** creates a new dictionary where each key is taken from the range 0 to 9, and the corresponding value is the square of that key. The resulting **squares** dictionary is then printed, showing the mapping of each number to its square.

6. Construct the set odd from the odd numbers in the range using a set comprehension (10).

Ans : Certainly! Here's an example of how you can use a set comprehension to create the set odd containing the odd numbers from the range 0 to 9 (range(10)):

odd = {num for num in range(10) if num % 2 != 0} print(odd)

Output:

{1, 3, 5, 7, 9}

In this example, the set comprehension **{num for num in range(10) if num % 2 != 0}** creates a new set containing only the odd numbers from the range of 0 to 9. The condition **if num % 2 != 0** checks if the number is not divisible by 2, indicating it's an odd number. The resulting set **{1, 3, 5, 7, 9}** is then printed.

7. Use a generator comprehension to return the string 'Got ' and a number for the numbers in range(10). Iterate through this by using a for loop.

Ans : Certainly! Here's an example of how you can use a generator comprehension to create a generator that yields the string 'Got ' followed by a number from the range 0 to 9 (range(10)). Then, you can iterate through this generator using a for loop:

generator = ('Got ' + str(num) for num in range(10)) for item in generator: print(item)

Output:

Got 0 Got 1 Got 2 Got 3 Got 4 Got 5 Got 6 Got 7 Got 8 Got 9

In this example, the generator comprehension **('Got ' + str(num) for num in range(10))** creates a generator that yields the string 'Got ' concatenated with each number from the range 0 to 9. The generator is then iterated through using a for loop, and each item is printed on a separate line.

8. Define a function called good that returns the list ['Harry', 'Ron', 'Hermione'].

Ans : Certainly! Here's an example of how you can define a function called good that returns the list ['Harry', 'Ron', 'Hermione']:

def good(): return ['Harry', 'Ron', 'Hermione'] # Call the function and store the result result = good() print(result)

Output:

['Harry', 'Ron', 'Hermione']

In this example, the function **good** is defined with no arguments. It simply uses the **return** statement to return the list **['Harry', 'Ron', 'Hermione']**. When the function is called and its result is stored in the variable **result**, the list **['Harry', 'Ron', 'Hermione']** is printed.

9. Define a generator function called get\_odds that returns the odd numbers from range(10). Use a for loop to find and print the third value returned.

Ans : Certainly! Here's an example of how you can define a generator function called get\_odds that yields the odd numbers from the range 0 to 9 (range(10)). Then, you can use a for loop to find and print the third value returned by the generator:

def get\_odds(): for num in range(10): if num % 2 != 0: yield num # Use a for loop to find and print the third value for index, value in enumerate(get\_odds()): if index == 2: print(value) break

Output:

5

In this example, the generator function **get\_odds** is defined. It iterates through the range 0 to 9, and for each number that is odd (i.e., not divisible by 2), it yields the number using the **yield** keyword.

Next, a for loop is used to iterate over the values generated by **get\_odds()**. The **enumerate()** function is used to get the index and value of each iteration. When the index is equal to 2, indicating the third value, it is printed using **print(value)**. Finally, the loop is exited using **break** to only print the third value.

10. Define an exception called OopsException. Raise this exception to see what happens. Then write the code to catch this exception and print 'Caught an oops'.

Ans : Certainly! Here's an example of how you can define an exception called OopsException, raise the exception, and then catch it using a try-except block to print 'Caught an oops':

# Define the OopsException class OopsException(Exception): pass try: # Raise the OopsException raise OopsException except OopsException: print('Caught an oops')

Output:

Caught an oops

In this example, the **OopsException** is defined as a custom exception by creating a class that inherits from the base **Exception** class. The **pass** statement indicates that the exception doesn't have any additional implementation.

Within the try block, the **raise** statement is used to explicitly raise the **OopsException**. Then, the except block catches the **OopsException** and the code within the block is executed, which in this case prints 'Caught an oops' to the console.

11. Use zip() to make a dictionary called movies that pairs these lists: titles = ['Creature of Habit', 'Crewel Fate'] and plots = ['A nun turns into a monster', 'A haunted yarn shop'].

Ans : Certainly! Here's an example of how you can use the zip() function to create a dictionary called movies by pairing the titles and plots lists:

titles = ['Creature of Habit', 'Crewel Fate'] plots = ['A nun turns into a monster', 'A haunted yarn shop'] movies = dict(zip(titles, plots)) print(movies)

Output:

{'Creature of Habit': 'A nun turns into a monster', 'Crewel Fate': 'A haunted yarn shop'}

In this example, the **zip()** function is used to pair corresponding elements from the **titles** and **plots** lists together. The **dict()** function is then used to convert the resulting zip object into a dictionary. The **movies** dictionary is printed, showing the pairing of titles with their respective plots.