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: code to accomplish:

# Assigning the value 7 to the variable guess\_me

guess\_me = 7

# Conditional tests using if, else, and elif

if guess\_me < 7:

print('too low')

elif guess\_me > 7:

print('too high')

else:

print('just right')

This code will correctly print 'too low' if `guess\_me` is less than 7, 'too high' if it's greater than 7, and 'just right' if it's equal to 7.

1. 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: The Python code that includes the while loop with the specified conditions:

# Assigning the value 7 to the variable guess\_me and 1 to the variable start

guess\_me = 7

start = 1

# While loop to compare start with guess\_me

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

When you run this code, it will compare the value of `start` with `guess\_me` in each iteration of the while loop. If `start` is less than `guess\_me`, it will print 'too low'. If `start` is equal to `guess\_me`, it will print 'found it!' and exit the loop. If `start` is greater than `guess\_me`, it will print 'oops' and also exit the loop. The `start` variable is incremented at the end of each loop iteration.

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

Ans: The Python code to print the values of the list [3, 2, 1, 0] using a for loop:

# The list [3, 2, 1, 0]

my\_list = [3, 2, 1, 0]

# For loop to print the values

for value in my\_list:

print(value)

When you run this code, it will iterate through the elements of the list `my\_list` and print each value on a separate line:

3

2

1

0

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

Ans: List of comprehension to create a list of the even numbers in the range from 0 to 9 (i.e., range(10)). Here's how you can do it in Python:

# List comprehension to generate a list of even numbers in range(10)

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

# Print the list of even numbers

print(even\_numbers)

When you run this code, it will output:

[0, 2, 4, 6, 8]

The list comprehension iterates through the numbers in the range from 0 to 9, and it only includes the numbers that are even (i.e., those that have a remainder of 0 when divided by 2) in the new list `even\_numbers`.

1. 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: We can use a dictionary comprehension to create the `squares` dictionary, where the keys are the numbers from 0 to 9 (using `range(10)`) and the values are the squares of each key. Here's how you can do it in Python:

# Dictionary comprehension to create the squares dictionary

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

# Print the squares dictionary

print(squares)

When you run this code, it will output:

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

The dictionary comprehension iterates through the numbers in the range from 0 to 9, using each number as both the key and the value, with the value being the square of the key. This way, it generates a dictionary containing the squares of the numbers from 0 to 9.

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

Ans: We can use a set comprehension to create the `odd` set containing the odd numbers from the range 0 to 9. Here's how you can do it in Python:

# Set comprehension to create the odd set

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

# Print the odd set

print(odd)

When you run this code, it will output:

{1, 3, 5, 7, 9}

The set comprehension iterates through the numbers in the range from 0 to 9, and it includes only the numbers that are odd (i.e., those that have a remainder of 1 when divided by 2) in the new set `odd`. This way, it generates a set containing the odd numbers from 0 to 9.

1. 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: We can use a generator comprehension to generate a series of strings like 'Got 0', 'Got 1', ..., 'Got 9' for the numbers in the range from 0 to 9. Then, you can iterate through this generator using a for loop. Here's how you can do it in Python:

# Generator comprehension to generate the strings

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

# Iterating through the generator using a for loop

for item in generator:

print(item)

When you run this code, it will output:

Got 0

Got 1

Got 2

Got 3

Got 4

Got 5

Got 6

Got 7

Got 8

Got 9

The generator comprehension creates a generator that yields the strings 'Got 0', 'Got 1', ..., 'Got 9' for the numbers in the range from 0 to 9. The for loop then iterates through this generator and prints each string on a separate line. The use of a generator is memory-efficient, as it generates the values on-the-fly and doesn't store them all in memory at once.

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

ANS: Here's the Python code to define the function `good` that returns the list `['Harry', 'Ron', 'Hermione']`:

def good():

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

# Call the function and store the returned list

result = good()

# Print the returned list

print(result)

When you run this code, it will output:

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

The function `good` simply returns the list `['Harry', 'Ron', 'Hermione']`, and then we call the function to retrieve the list and print it.

1. 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: Here's the Python code to define the generator function `get\_odds` that yields the odd numbers from `range(10)` and then use a for loop to find and print the third value returned:

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 returned by the generator

count = 0

for value in get\_odds():

count += 1

if count == 3:

print("Third odd number:", value)

break

When you run this code, it will output:

Third odd number: 5

The `get\_odds` generator function uses a for loop to iterate through the numbers in `range(10)`, and it yields only the odd numbers (i.e., those that have a remainder of 1 when divided by 2). The for loop then iterates through the generator until it reaches the third value, at which point it prints that value and breaks out of the loop. This way, it finds and prints the third odd number returned by the generator.

1. 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: Here's the Python code to define the `OopsException`, raise the exception, and then catch it using a try-except block to print 'Caught an oops':

# Define the custom exception OopsException

class OopsException(Exception):

pass

try:

# Raise the OopsException

raise OopsException("Something went wrong!")

except OopsException as e:

# Catch the exception and print the message

print('Caught an oops:', e)

When you run this code, it will output:

Caught an oops: Something went wrong!

In this code, we define the custom exception `OopsException` by creating a new class that inherits from the built-in `Exception` class. Then, we use a try-except block to raise the `OopsException` and catch it when it occurs. The catch block prints the message associated with the exception.

1. 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: We can use the `zip()` function to pair the `titles` and `plots` lists and create the `movies` dictionary. Here's how you can do it in Python:

# Lists of titles and plots

titles = ['Creature of Habit', 'Crewel Fate']

plots = ['A nun turns into a monster', 'A haunted yarn shop']

# Using zip() to create the movies dictionary

movies = dict(zip(titles, plots))

# Print the movies dictionary

print(movies)

When you run this code, it will output:

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

The `zip()` function combines the corresponding elements of `titles` and `plots` into pairs, and then the `dict()` function creates a dictionary from those pairs. The resulting `movies` dictionary will have the titles as keys and the plots as values.