**Assignment 17**

**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.**

**Here's the solution to your problem:**

**python**

**Copy code**

**guess\_me = 7**

**if guess\_me < 7:**

**print('too low')**

**elif guess\_me > 7:**

**print('too high')**

**else:**

**print('just right')**

**In the above code, we assign the value of 7 to the variable guess\_me. Then, we use if, elif, and else conditional statements to check if the value of guess\_me is less than, greater than, or equal to 7. If guess\_me is less than 7, we print 'too low'. If guess\_me is greater than 7, we print 'too high'. If guess\_me is equal to 7, we print '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.**

**Here's the solution to your problem:**

**python**

**Copy code**

**guess\_me = 7**

**start = 1**

**while True:**

**if start < guess\_me:**

**print('too low')**

**elif start == guess\_me:**

**print('found it!')**

**break**

**else:**

**print('oops')**

**break**

**start += 1**

**In the above code, we assign the value of 7 to the variable guess\_me and the value of 1 to the variable start. We then use a while loop to compare start with guess\_me. If start is less than guess\_me, we print 'too low'. If start is equal to guess\_me, we print 'found it!' and exit the loop using the break statement. If start is greater than guess\_me, we print 'oops' and exit the loop using the break statement. At the end of each iteration of the loop, we increment the value of start by 1 using the += operator.**

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

**Here's the solution to your problem:**

**python**

**Copy code**

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

**for value in my\_list:**

**print(value)**

**In the above code, we assign the list [3, 2, 1, 0] to the variable my\_list. We then use a for loop to iterate over the values in my\_list and print each value on a separate line. The loop will execute four times, printing the values 3, 2, 1, and 0.**

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

**Here's the solution to your problem:**

**python**

**Copy code**

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

**print(even\_numbers)**

**In the above code, we use a list comprehension to create a list of even numbers in the range from 0 to 9. We iterate over each number in the range using range(10), and include it in the list only if it is even, i.e., if its remainder when divided by 2 is zero, using the condition if num % 2 == 0. Finally, we print the resulting list of even numbers, which is [0, 2, 4, 6, 8].**

**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.**

**Here's the solution to your problem:**

**python**

**Copy code**

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

**print(squares)**

**In the above code, we use a dictionary comprehension to create a dictionary named squares. We iterate over each number in the range from 0 to 9 using range(10), and set the key to be the number itself, and the value to be its square, using the syntax num: num\*\*2. Finally, we print the resulting dictionary squares. It will have the following key-value pairs:**

**yaml**

**Copy code**

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

**Each key is a number from 0 to 9, and each corresponding value is the square of that number.**

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

**Here's the solution to your problem:**

**python**

**Copy code**

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

**print(odd)**

**In the above code, we use a set comprehension to create a set named odd that contains only odd numbers in the range from 0 to 9. We iterate over each number in the range using range(10), and include it in the set only if it is odd, i.e., if its remainder when divided by 2 is not zero, using the condition if num % 2 != 0. Finally, we print the resulting set odd. It will contain the following odd numbers:**

**Copy code**

**{1, 3, 5, 7, 9}**

**Each number in the set is odd, and no even numbers are included.**

**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.**

**Here's the solution to your problem:**

**python**

**Copy code**

**gen\_comp = ('Got ' + str(num) for num in range(10))**

**for item in gen\_comp:**

**print(item)**

**In the above code, we use a generator comprehension to create a generator expression that yields the string 'Got ' followed by each number in the range from 0 to 9, converted to a string. We then use a for loop to iterate through the items generated by the generator expression, and print each item on a separate line. The loop will execute ten times, printing the strings 'Got 0', 'Got 1', 'Got 2', and so on, up to 'Got 9'.**

**Note that a generator expression is like a list comprehension, but it generates items on-the-fly as they are needed, rather than creating a list of all items at once. This can be more memory-efficient when working with large data sets.**

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

**Here's the solution to your problem:**

**python**

**Copy code**

**def good():**

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

**In the above code, we define a function named good that takes no arguments and simply returns a list of strings ['Harry', 'Ron', 'Hermione']. When the function is called, it will generate and return this list, which can then be used elsewhere in the program.**

**To use this function, simply call it by its name followed by parentheses, like so: good(). This will execute the function and return the list [‘Harry’, ‘Ron’, ‘Hermione’].**

**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.**

**Here's the solution to your problem:**

**python**

**Copy code**

**def get\_odds():**

**for num in range(1, 10, 2):**

**yield num**

**count = 1**

**for num in get\_odds():**

**if count == 3:**

**print("The third odd number is:", num)**

**break**

**count += 1**

**In the above code, we define a generator function called get\_odds that yields odd numbers from the range 1 to 9 (inclusive). We use the range() function to generate all odd numbers in that range with a step of 2.**

**Next, we use a for loop to iterate over the values generated by the get\_odds() generator function. We use a counter variable count to keep track of the number of values we've seen so far. When we reach the third value (i.e., when count == 3), we print a message indicating that we've found the third value, along with the value itself. We then break out of the loop using the break statement.**

**When you run this code, it will output the following message:**

**csharp**

**Copy code**

**The third odd number is: 5**

**This is because the third odd number in the range 1 to 9 is 5.**

**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’.**

**Here's the solution to your problem:**

**python**

**Copy code**

**# Define an exception called OopsException**

**class OopsException(Exception):**

**pass**

**# Raise the exception to see what happens**

**raise OopsException**

**# Use try/except to catch the exception and print a message**

**try:**

**raise OopsException**

**except OopsException:**

**print('Caught an oops')**

**In the above code, we define a custom exception called OopsException by creating a new class that inherits from the built-in Exception class. We then raise this exception using the raise statement, which will cause the program to terminate and display a stack trace showing the location where the exception was raised.**

**We then use a try/except block to catch the OopsException and print a message indicating that we caught it. When the exception is raised, the program flow will transfer to the except block, which will execute the code within it (in this case, printing the message 'Caught an oops').**

**When you run this code, it will output the following message:**

**Copy code**

**Caught an oops**

**This indicates that the OopsException was caught and handled properly by the try/except block.**

**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’].**

**Here's the solution to your problem:**

**python**

**Copy code**

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

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

**# Use zip() to combine the titles and plots into a dictionary called movies**

**movies = dict(zip(titles, plots))**

**# Print the resulting dictionary**

**print(movies)**

**In the above code, we use the built-in zip() function to combine the two lists titles and plots into a dictionary called movies. The zip() function creates an iterator that aggregates elements from each of the input iterables. In this case, it returns a series of tuples that pair up the corresponding elements from titles and plots, which we then convert to a dictionary using the dict() function.**

**When you run this code, it will output the following dictionary:**

**arduino**

**Copy code**

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

**This is the movies dictionary we constructed by pairing the titles and plots lists.**