Name :- Deepawali . B. Mhaisagar
Assignment no 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.

ANS

Here's an example of how to assign the value 7 to the variable `guess_me` and write the conditional tests using `if`, `else`, and `elif` statements to print the corresponding string based on the value of `guess_me`:

```
guess_me = 7

if guess_me < 7:

    print('too low')

elif guess_me > 7:

    print('too high')

else:

    print('just right')
```

In this code, the value 7 is assigned to the variable `guess_me`. The conditional tests are written using `if`, `elif` (short for "else if"), and `else` statements

If `guess_me` is less than 7, the condition `guess_me < 7` evaluates to `True`, and the string `'too low'` is printed. If `guess_me` is greater than 7, the condition `guess_me > 7` evaluates to `True`, and the string `'too high'` is printed. Finally, if `guess_me` is equal to 7, none of the previous conditions are met, and the string `'just right'` is printed.

Depending on the value of `guess_me`, one of the three strings will be printed: `'too low'`, `'too high'`, or `'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

Here's an example of how to use a while loop to compare the variables `start` and `guess_me`, printing the corresponding messages based on their values:

```
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
```

In this code, the value 7 is assigned to the variable `guess_me`, and the value 1 is assigned to the variable `start`. The while loop continues as long as `start` is less than or equal to `guess_me`.

Inside the loop, it checks the conditions using if, elif, and else statements. 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 exits the loop using the `break` statement.

At the end of each iteration, the value of `start` is incremented by 1 using `start += 1`.

The output of the loop will depend on the values of `guess_me` and `start`, printing the appropriate messages based on the comparisons made.

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

ANS

To print the values of the list `[3, 2, 1, 0]` using a for loop, you can iterate over the elements of the list and print each element. Here's an example:

```
my_list = [3, 2, 1, 0]



for num in my_list:

    print(num)
```

After executing this code, the output will be:

```
3

2

1

0
```

The for loop iterates over each element in the list `my_list` and assigns it to the variable `num`. The `print(num)` statement then prints each element on a separate line.

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

ANS

To create a list of even numbers in the range from 0 to 9 (range(10)) using a list comprehension, you can specify the condition that filters out odd numbers. Here's an example:

```
even_numbers = [num for num in range(10) if num % 2 == 0]

print(even_numbers)
```

After executing this code, the output will be:

```
[0, 2, 4, 6, 8]
```

In this code, the list comprehension `[num for num in range(10) if num % 2 == 0]` generates a new list by iterating over the numbers in the range from 0 to 9 (exclusive). The condition `num % 2 == 0` filters out the odd numbers by checking if the number is divisible by 2 without a remainder (i.e., an even number). Only the even numbers satisfy the condition and are included in the resulting list `even_numbers`.

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

To create a dictionary called `squares` using a dictionary comprehension, where the keys are the numbers from 0 to 9 (using `range(10)`) and the values are the squares of each key, you can use the following code:

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

print(squares)
```

After executing this code, the output will be:

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

In this code, the dictionary comprehension `{num: num**2 for num in range(10)}` generates a new dictionary by iterating over the numbers in the range from 0 to 9. For each number `num`, the key-value pair is created, where the key is `num` and the value is the square of `num` (`num**2`). The resulting dictionary `squares` contains the numbers from 0 to 9 as keys, with their corresponding squared values as the respective values.

6. Make a surprise list with the elements “Groucho,”; “Chico”; and “Harpo”

ANS

To construct a set called `odd` containing the odd numbers from the range 0 to 9 (using `range(10)`) using a set comprehension, you can use the following code:

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

print(odd)
```

After executing this code, the output will be:

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

In this code, the set comprehension `{num for num in range(10) if num % 2 != 0}` generates a new set by iterating over the numbers in the range from 0 to 9. The condition `num % 2 != 0` filters out the even numbers by checking if the number is not divisible by 2 (i.e., an odd number). Only the odd numbers satisfy the condition and are included in the resulting set `odd`.

7. Use a generator comprehension to return the string &#39;Got &#39; and a number for the numbers in range(10). Iterate through this by using a for loop.

ANS

To use a generator comprehension to return the string `'Got '` followed by a number for each number in the range from 0 to 9 (using `range(10)`), and then iterate through the generated values using a for loop, you can use the following code:

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



for item in generator:

    print(item)
```

After executing this code, the output will be:

```
Got 0

Got 1

Got 2

Got 3

Got 4
Got 5

Got 6

Got 7

Got 8

Got 9
```

In this code, the generator comprehension `('Got ' + str(num) for num in range(10))` generates a sequence of strings by concatenating the string `'Got '` with each number in the range from 0 to 9. The generator does not generate all the values at once but rather generates them on-the-fly as requested.

The for loop then iterates through the generated values and prints each item in the sequence. Each item is a string consisting of `'Got '` followed by the corresponding number from the range.

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

ANS

To define a function called `good` that returns the list `['Harry', 'Ron', 'Hermione']`, you can use the following code:

```
def good():

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



# Example usage

result = good()

print(result)
```

After executing this code, the output will be:

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

In this code, the `good` function is defined using the `def` keyword. The function does not take any arguments and simply returns the list `['Harry', 'Ron', 'Hermione']` using the `return` statement.

To use the `good` function, you can call it and store the returned value in a variable (in this case, `result`) and then print the value. The result will be 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.

ANS

To define a generator function called `get_odds` that returns the odd numbers from the range 0 to 9 (using `range(10)`), and then use a for loop to find and print the third value returned, you can use the following code:

```
def get_odds():

    for num in range(10):

        if num % 2 != 0:

            yield num



# Example usage

count = 0

for odd in get_odds():

    count += 1

    if count == 3:

        print(odd)

        break
```

After executing this code, the output will be:

```
5
```

In this code, the `get_odds` function is defined as a generator function using the `def` keyword. It iterates through the numbers in the range from 0 to 9 and yields only the odd numbers using the condition `num % 2 != 0`. The `yield` statement allows the function to generate the next odd number in each iteration.

To find and print the third value returned by the generator function, a variable `count` is used to keep track of the count. The for loop iterates over the generator and increments `count` for each odd number encountered. When `count` reaches 3, the third odd number is printed using `print(odd)` and the loop is exited using the `break` statement. In this case, the third odd number returned is 5.

10. Make a French-to-English dictionary called f2e from e2f. Use the items method.

ANS

To define an exception called `OopsException`, raise this exception to see what happens, and then write the code to catch this exception and print `'Caught an oops'`, you can use the following code:

```
class OopsException(Exception):

    pass



try:

    raise OopsException

except OopsException:

    print('Caught an oops')
```

After executing this code, the output will be:

```
Caught an oops
```

In this code, a custom exception called `OopsException` is defined by creating a new class that inherits from the built-in `Exception` class. The `pass` statement is used to indicate that the class does not contain any additional methods or properties.

Within the `try` block, the `raise` statement is used to raise an instance of the `OopsException` exception. This causes an exception to be raised at that point in the code.

The `except` block then catches the `OopsException` exception and executes the code within it, which prints the message `'Caught an oops'`. This block is only executed if the corresponding exception is caught.

By running this code, you will see the message `'Caught an oops'` printed, indicating that the exception was caught and the corresponding code block was executed.

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

To use the `zip()` function to create a dictionary called `movies` that pairs the lists `titles` and `plots`, you can combine them as key-value pairs using `zip()` and convert the result into a dictionary. Here's an example:

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

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

movies = dict(zip(titles, plots))

print(movies)
```

After executing this code, the output will be:

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

In this code, `zip(titles, plots)` combines the elements of the `titles` and `plots` lists into pairs. The `dict()` function then converts these pairs into a dictionary, creating key-value pairs where each title is associated with its corresponding plot.

The resulting `movies` dictionary contains the movies' titles as keys and their plots as the respective values.