**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.**

Sure, here's the Python code for the given task:

```python

# Assign 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')

```

When you run this code, it will output `'just right'` since `guess\_me` is equal to 7. If you change the value of `guess\_me` to any other number, the code will print `'too low'` if the value is less than 7 or `'too high'` if the value is greater than 7.

**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.**

Certainly! Here's the Python code for the while loop with the described conditions:

```python

# Assign the values 7 to guess\_me and 1 to 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 output:

```

too low

too low

too low

too low

too low

too low

found it!

```

The loop runs until `start` reaches the value of `guess\_me`, and when they are equal, it prints `'found it!'`. If `start` becomes greater than `guess\_me`, it will print `'oops'` and exit the loop.

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

Sure, you can use a for loop to print the values of the list [3, 2, 1, 0] like this:

```python

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

for value in my\_list:

print(value)

```

When you run this code, it will output:

```

3

2

1

0

```

The for loop iterates over each element in the list and prints them one by one.

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

Sure! You can use a list comprehension to create a list of even numbers in the range from 0 to 9 (range(10)) like this:

```python

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

print(even\_numbers)

```

When you run this code, it will output:

```

[0, 2, 4, 6, 8]

```

The list comprehension iterates over each number in the range from 0 to 9, and for each number, it checks if it is even (divisible by 2 with a remainder of 0). If it is even, the number is included in the `even\_numbers` list.

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.**

Certainly! You can use a dictionary comprehension to create the dictionary `squares`, where keys are taken from the `range(10)` and the corresponding values are the squares of each key. Here's how you can do it:

```python

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

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 over each number in the range from 0 to 9, and for each number, it creates a key-value pair with the key being the number itself and the value being the square of that number.

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

Sure! You can use a set comprehension to create the set `odd` containing the odd numbers in the range from 0 to 9 (range(10)). Here's how you can do it:

```python

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

print(odd)

```

When you run this code, it will output:

```

{1, 3, 5, 7, 9}

```

The set comprehension iterates over each number in the range from 0 to 9, and for each number, it checks if it is odd (not divisible by 2 with a remainder of 0). If it is odd, the number is included in the `odd` set.

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.**

Sure! You can use a generator comprehension to generate a sequence of strings `'Got '` followed by each number in the range from 0 to 9 (range(10)). Then, you can iterate through this generator using a for loop. Here's how you can do it:

```python

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

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 generates a sequence of strings using the formula `'Got ' + str(num)` for each number in the range from 0 to 9. Then, the for loop iterates through the generator and prints each item in the sequence.

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

Sure! Here's the Python function `good` that returns the list `['Harry', 'Ron', 'Hermione']`:

```python

def good():

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

# Call the function and store the result in a variable

result\_list = good()

# Print the result

print(result\_list)

```

When you run this code, it will output:

```

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

```

The `good` function simply returns the list `['Harry', 'Ron', 'Hermione']`. When you call the function and store the result in a variable (`result\_list` in this case), you can access and work with the returned list as needed.

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.**

Sure! Here's the Python generator function `get\_odds` that returns the odd numbers from the range 0 to 9 (range(10)):

```python

def get\_odds():

for num in range(10):

if num % 2 != 0:

yield num

# Using a for loop to find and print the third value returned

count = 0

for odd\_num in get\_odds():

count += 1

if count == 3:

print("Third odd number:", odd\_num)

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 the range from 0 to 9. It checks if each number is odd (not divisible by 2 with a remainder of 0) and yields the odd numbers. The for loop iterates through the generator until it finds the third odd number, which is 5 in this case, and then it prints that number.

**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’.**

Sure! Let's define the `OopsException` exception and then raise it to see what happens. After that, we'll catch the exception using a try-except block and print the message "Caught an oops". Here's the code:

```python

# Define the 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!

```

The `OopsException` is a custom exception class that we defined, and when we raise it using `raise OopsException(...)`, the program enters the exception state. The try-except block catches the `OopsException` and prints the message specified in the `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’].**

Certainly! You can use the `zip()` function to create a dictionary called `movies` by pairing the lists `titles` and `plots`. Here's the code:

```python

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

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

# Using zip() to create the dictionary movies

movies = dict(zip(titles, plots))

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 elements from the two lists `titles` and `plots` pairwise and then, the `dict()` function creates a dictionary from those pairs. The resulting dictionary `movies` contains the titles as keys and the corresponding plots as values.