### Q1. Creating a Function to Return a List of Odd Numbers in the Range of 1 to 25

In Python, the keyword used to create a function is def.

Here's a function to return a list of odd numbers in the range of 1 to 25:

python

Copy code

def get\_odd\_numbers():

return [i for i in range(1, 26) if i % 2 != 0]

print(get\_odd\_numbers())

### Q2. Using \*args and \*\*kwargs

\*args and \*\*kwargs are used to pass a variable number of arguments to a function. \*args is used to pass a non-keyworded variable-length argument list, while \*\*kwargs allows you to pass keyworded variable-length arguments.

Here's an example function using \*args:

python

Copy code

def sum\_all(\*args):

return sum(args)

print(sum\_all(1, 2, 3, 4, 5))

Here's an example function using \*\*kwargs:

python

Copy code

def print\_info(\*\*kwargs):

for key, value in kwargs.items():

print(f"{key}: {value}")

print\_info(name="John", age=25, city="New York")

### Q3. Using Iterators in Python

An iterator in Python is an object that contains a countable number of values. The \_\_iter\_\_() method initializes the iterator object, and the \_\_next\_\_() method is used for iteration.

Here's how you can use these methods to print the first five elements of the given list [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]:

python

Copy code

numbers = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

iterator = iter(numbers)

for \_ in range(5):

print(next(iterator))

### Q4. Generator Function in Python

A generator function in Python is a function that returns an iterator that produces a sequence of values when iterated over. The yield keyword is used to produce a value and suspend the function’s state.

Here's an example of a generator function:

python

Copy code

def simple\_generator():

for i in range(3):

yield i

gen = simple\_generator()

for value in gen:

print(value)

### Q5. Generator Function for Prime Numbers Less Than 1000

Here's a generator function to produce prime numbers less than 1000:

python

Copy code

def prime\_numbers():

def is\_prime(n):

if n < 2:

return False

for i in range(2, int(n\*\*0.5) + 1):

if n % i == 0:

return False

return True

num = 2

while num < 1000:

if is\_prime(num):

yield num

num += 1

prime\_gen = prime\_numbers()

for \_ in range(20):

print(next(prime\_gen))

### Q6. First 10 Fibonacci Numbers Using a While Loop

Here's a program to print the first 10 Fibonacci numbers using a while loop:

python

Copy code

def fibonacci(n):

a, b = 0, 1

count = 0

while count < n:

print(a)

a, b = b, a + b

count += 1

fibonacci(10)

### Q7. List Comprehension to Iterate Through the Given String 'pwskills'

Here's a list comprehension to iterate through the string 'pwskills':

python

Copy code

string = 'pwskills'

output = [char for char in string if char in 'pwskills']

print(output)

### Q8. Checking if a Number is a Palindrome Using a While Loop

Here's a program to check if a given number is a palindrome:

python

Copy code

def is\_palindrome(num):

original\_num = num

reversed\_num = 0

while num > 0:

reversed\_num = reversed\_num \* 10 + num % 10

num = num // 10

return original\_num == reversed\_num

number = 121 # Replace with the number you want to check

print(is\_palindrome(number))

### Q9. Printing Odd Numbers from 1 to 100 Using List Comprehension

Here's the code to print odd numbers from 1 to 100 using list comprehension:

python

Copy code

numbers = [i for i in range(1, 101)]

odd\_numbers = [num for num in numbers if num % 2 != 0]

print(odd\_numbers)