Q1. Which keyword is used to create a function?

The keyword `def` is used to create a function in Python.

Example: Create a function to return a list of odd numbers from 1 to 25

```
def odd_numbers():
    return [num for num in range(1, 26) if num % 2 != 0]

# Test the function
print(odd_numbers())
```

Output
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25]

Q2. Why `*args` and `**kwargs` are used?

- `*args` allows a function to accept **any number of positional arguments**.
- `**kwargs` allows a function to accept **any number of keyword arguments**.

Example using `*args`:

```
def sum_numbers(*args):
    return sum(args)

print(sum_numbers(1, 2, 3, 4, 5))  # 15

def display_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

display_info(name="Alice", age=25, city="New York")
```

Output
name: Alice
age: 25
city: New York

Q3. What is an iterator in Python?

- An **iterator** is an object that allows you to traverse through all the elements of a collection (like a list) **one at a time**.
- To create and use an iterator:
    - `iter()` → initializes an iterator object
    - `next()` → retrieves the next item

Example: Print first five elements of `[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]`

```
numbers = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
it = iter(numbers)  # initialize iterator

# Print first 5 elements
for _ in range(5):
    print(next(it))
```

Output
2
4
6
8
10

Q4. What is a generator function? Why `yield` is used?

- A **generator function** returns an **iterator** that yields items **one at a time** instead of returning the entire list at once.
- `yield` is used to **pause** the function and return a value, resuming from the same point on the next call.

Example of a generator function:

```
def squares(n):
    for i in range(1, n+1):
        yield i**2

gen = squares(5)
for val in gen:
```

```
    print(val)
```

Output:

```
1
4
9
16
25
```

Q5. Generator function for prime numbers less than 1000

```
def prime_generator(limit=1000):
    for num in range(2, limit):
        for i in range(2, int(num**0.5)+1):
            if num % i == 0:
                break
        else:
            yield num

# Create generator
primes = prime_generator()

# Print first 20 prime numbers
for _ in range(20):
    print(next(primes))
```

Output (first 20 primes):

```
2
3
5
7
11
13
17
19
```

23
29
31
37
41
43
47
53
59
61
67
71
