Functions:

- Functions are blocks of reusable code that can be called multiple times in a program.
- They are defined using the `def` keyword, followed by the function name, parameters within parentheses and a colon.
- The code inside the function is indented and executed when the function is called.
- Functions can return values using the `return` keyword. If a function does not have a return statement, it returns `None` by default.

```
def greet(name):
    """This function greets the person passed in as a parameter"""
    print("Hello, " + name + ". How are you today?")

greet("John")
# Output: Hello, John. How are you today?
```

Arguments:

- Functions can take arguments as inputs.
- There are different types of arguments:
  - Positional arguments: passed to the function in the same order as they are defined in the function.
  - Keyword arguments: passed to the function using the argument name and value.
  - Default arguments: assigned a default value in the function definition. If no value is provided during the function call, the default value is used.
  - Variable-length arguments: the number of arguments can be varied.

```
def greet(name, msg="Good morning!"):
    """This function greets the person passed in as a parameter"""
    print("Hello, " + name + ". " + msg)

greet("John", "How do you do?")
# Output: Hello, John. How do you do?

greet("John")
# Output: Hello, John. Good morning!
```

*args:

- It allows the function to take a variable number of positional arguments.
- `args` is used as a parameter in the function definition and collects all remaining positional arguments into a tuple.

```
def greet(*names):
    """This function greets all persons in the names tuple"""
    for name in names:
        print("Hello, " + name)

greet("John", "Jane", "Jim")
# Output:
# Hello, John
# Hello, Jane
# Hello, Jim
```

**kwargs:

- It allows the function to take a variable number of keyword arguments.

- `*kwargs` is used as a parameter in the function definition and collects all remaining keyword arguments into a dictionary.

```
def greet(**kwargs):
    """This function greets the person passed in as a parameter"""
    if kwargs:
        print("Hello, " + kwargs['name'] + ". " + kwargs['msg'])

greet(name="John", msg="How do you do?")
# Output: Hello, John. How do you do?
```

Generator Functions:

- A generator function is a special type of function that can be used to generate a sequence of values over time, instead of computing all the values at once and returning them in a list.
- The generator function uses the `yield` keyword instead of `return` to produce a value.
- Each time the generator function is called, it resumes execution from where it was paused and runs until it encounters the next `yield` statement.
- Generator functions are used for iteration, because they can generate values one by one, instead of all at once.

```
def my_range(n):
    """This is a generator function that yields values from 0 to n (exclusive)"""
    i = 0
    while i < n:
        yield i
        i += 1

for i in my_range(5):
    print(i)
# Output:
# 0
# 1
# 2
# 3
# 4
```