Lecture 9

Learning outcomes:

- *The general structure of Python programs*

❖What is a Program?

- A program, in the realm of programming languages of computer science and software development, is a definition of a set of instructions in a certain language which are given a task or tasked to resolve a particular problem.

❖What is the Python Programming Language?

- Python is a high-level, interpreted programming language that is easy to learn and use. It has a simple and easy-to-understand syntax that emphasizes readability and reduces the cost of program maintenance.

- Unlike many other programming languages, Python is well known for its simplicity and readability due to having a high-level interpretation that makes it not complicated to understand.

- Developed by Guido van Rossum in 1991 and released to the public, Python ranks among the most sought-after programming languages today, alongside Shell Script, Java, C++, and Perl, used in web development, data analysis, artificial intelligence, scientific computing, and more.

❖ ***The basic structure of a Python program consists of the following components:***

1. ***Comments:*** Comments are used to explain the purpose of the code or to make notes for other programmers. They start with a ‘#’ symbol and are ignored by the interpreter.

```
# This is a single-line comment

"""
This is a multi-line comment.
It can span multiple lines.
"""
```

2. Import Statements: Import statements are used to import modules or libraries into the program. These modules contain predefined functions that can be used to accomplish tasks.

```
import module_name
from module_name import function_name
```

3. Variable Assignments: Variables are used to store data in memory for later use. In Python, variables do not need to be declared with a specific type.

```
variable_name = 42
```

4. Data Types: Python supports several built-in data types including integers, floats, strings, booleans, and lists.

5. Operators: Operators are used to perform operations on variables and data. Python supports arithmetic, comparison, and logical operators.

6. Control Structures: Control structures are used to control the flow of a program. Python supports if-else statements, for loops, and while loops.

7. Functions: Functions are used to group a set of related statements together and give them a name. They can be reused throughout a program.

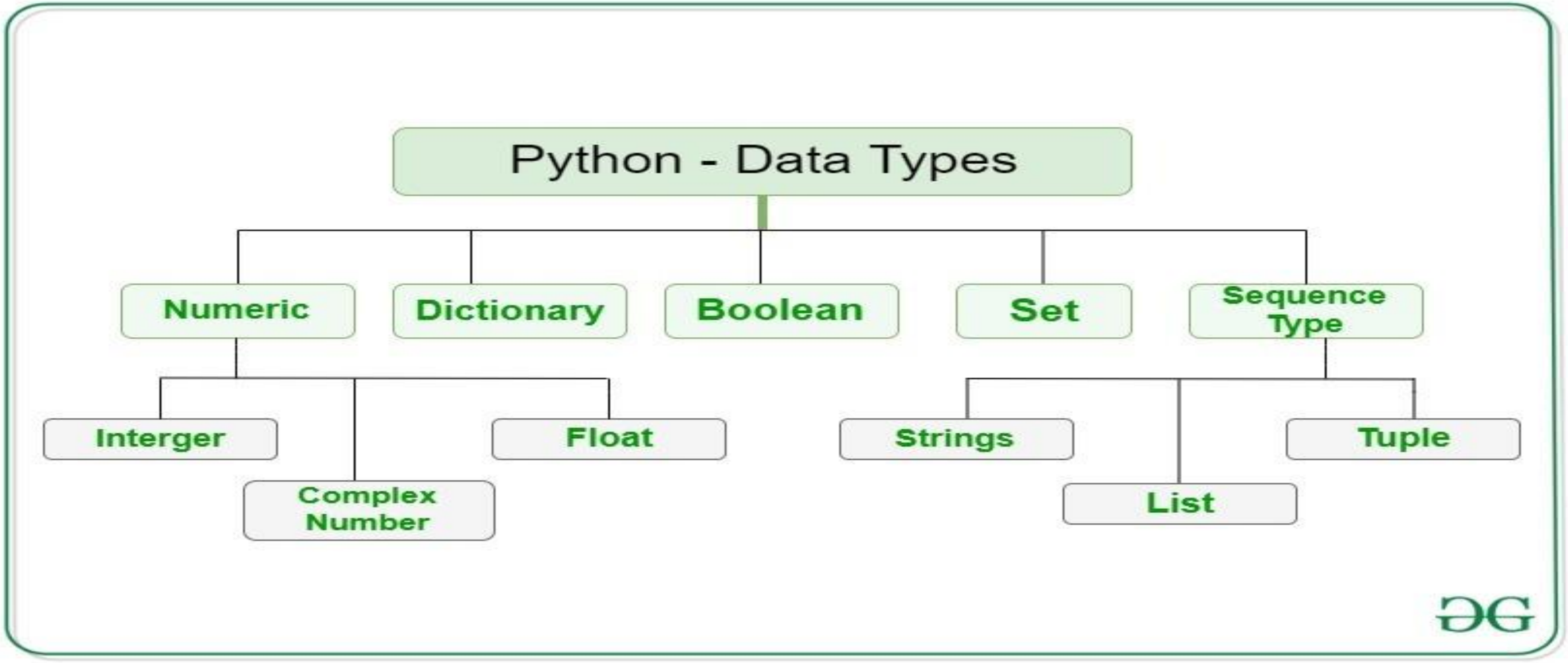
```
def my_function(parameter1, parameter2):
    # Function body
    result = parameter1 + parameter2
    return result
```

8. Classes: Classes are used to define objects that have specific attributes and methods. They are used to create more complex data structures and encapsulate code.

9. Exceptions: Exceptions are used to handle errors that may occur during the execution of a program.

Python Data Types

- Python offers, enabling you to manipulate and manage data with precision and flexibility.



Python Numbers

- In Python, “Numbers” is a category that encompasses different types of numeric data.
- Python supports various types of numbers, including integers, floating-point numbers, and complex numbers. Here’s a brief overview of each:

✓*Python Integer*

✓*Python Float*

✓*Python Complex*

✓*Type Conversion in Python*

✓*Decimal Numbers in Python*

Python Integer

- Python int is the whole number, including negative numbers but not fractions. In Python, there is no limit to how long an integer value can be.
- Example 1: Creating int and checking type

```
num = -8
# print the data type
print(type(num))
```

✓Output:

```
<class 'int'>
```

- Example 2: Performing arithmetic Operations on int type

```
a = 5
b = 6

# Addition
c = a + b
print("Addition:",c)
```

✓Output:

```
Addition: 11
```

```
d = 9
e = 6

# Subtraction
f = d - e
print("Subtraction:",f)
```

✓Output:

```
Subtraction: 3
```

```
g = 8
h = 2

# Division
i = g // h
print("Division:",i)
```

✓Output:

```
Division: 4
```

```
j = 3
k = 5

# Multiplication
l = j * k
print("Multiplication:",l)
```

✓Output:

```
Multiplication: 15
```

```
m = 25
n = 5

# Modulus
o = m % n
print("Modulus:",o)
```

✓Output:

```
Modulus: 0
```

```
p = 6
q = 2

# Exponent
r = p ** q
print("Exponent:",r)
```

✓Output:

```
Exponent: 36
```

Python Float

- This is a real number with a floating-point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation. Some examples of numbers that are represented as floats are 0.5 and -7.823457.

- They can be created directly by entering a number with a decimal point, or by using operations such as division on integers. Extra zeros present at the number’s end are ignored automatically.

- Example 1: Creating float and checking type

```
num = 3/4

# print the data type
print(type(num))
```

✓Output:

```
<class 'float'>
```

- As we have seen, dividing any two integers produces a float. A float is also produced by running an operation on two floats, or a float and an integer.

```
num = 6 * 7.0
print(type(num))
```

✓Output:

```
<class 'float'>
```

- Example 2: Performing arithmetic Operations on the float type

```
a = 5.5
b = 3.2

# Addition
c = a + b
print("Addition:", c)

# Subtraction
c = a-b
print("Subtraction:", c)

# Division
c = a/b
print("Division:", c)

# Multiplication
c = a*b
print("Multiplication:", c)
```

- ✓Output:

```
Addition: 8.7
Subtraction: 2.3
Division: 1.71875
Multiplication: 17.6
```

- o ***Note: The accuracy of a floating-point number is only up to 15 decimal places, the 16th place can be inaccurate.***

Python Complex

- A complex number is a number that consists of real and imaginary parts. For example, 2 + 3j is a complex number where 2 is the real component, and 3 multiplied by j is an imaginary part.
- Example 1: Creating Complex and checking type

```
num = 6 + 9j
print(type(num))
```

✓Output:

```
<class 'complex'>
```

- Example 2: Performing arithmetic operations on complex type

```
a = 1 + 5j
b = 2 + 3j

# Addition
c = a + b
print("Addition:",c)
```

✓Output:

```
Addition: (3+8j)
```

```
d = 1 + 5j
e = 2 - 3j

# Subtraction
f = d - e
print("Subtraction:",f)
```

✓Output:

```
Subtraction: (-1+8j)
```

```
j = 1 + 5j
k = 2 + 3j

# Multiplication
l = j * k
print("Multiplication:",l)
```

✓Output:

```
Multiplication: (-13+13j)
```

```
g = 1 + 5j
h = 2 + 3j

# Division
i = g / h
print("Division:",i)
```

✓Output:

```
Division: (1.307692307692308+0.5384615384615384j)
```

Type Conversion in Python

- We can convert one number into the other form by two methods:

1. Using Arithmetic Operations:

We can use operations like addition, and subtraction to change the type of number implicitly(automatically), if one of the operands is float. This method is not working for complex numbers.

➢*Example: Type conversion using arithmetic operations*

```
a = 1.6
b = 5
c = a + b
print(c)
```

✓Output:

```
6.6
```

2. *Using built-in functions*

We can also use built-in functions like int(), float() and complex() to convert into different types explicitly.

➢Example: Type conversion using built-in functions

```
a = 2
print(float(a))
```

✓Output:

```
2.0
```

```
b = 5.6
print(int(b))
```

✓Output:

```
5
```

```
c = '3'
print(type(int(c)))
```

✓Output:

```
<class 'int'>
```

```
d = '5.6'
print(type(float(d)))
```

✓Output:

```
<class 'float'>
```

```
e = 5
print(complex(e))
```

✓Output:

```
(5+0j)
```

```
f = 6.5
print(complex(f))
```

✓Output:

```
(6.5+0j)
```

- When we convert float to int, the decimal part is truncated.

- ***Note:***

1. We can’t convert a complex data type number into int data type and float data type numbers.
2. We can’t apply complex built-in functions on strings.

Decimal Numbers in Python

- Arithmetic operations on the floating number can give some unexpected results.
- Example 1: Let’s consider a case where we want to add 1.1 to 2.2. You all must be wondering why the result of this operation should be 3.3 but let’s see the output given by Python.

```
a = 1.1
b = 2.2
c = a+b
print(c)
```

✓Output:

```
3.3000000000000003
```

- Example 2: The result can be unexpected. Let’s consider another case where we will subtract 1.2 and 1.0. Again we will expect the result as 0.2, but let’s see the output given by Python.

```
a = 1.2
b = 1.0
c = a-b
print(c)
```

✓Output:

```
0.19999999999999996
```

String

What is a String in Python?

- Python Strings are arrays of bytes representing Unicode characters. In simpler terms, a string is an immutable array of characters.

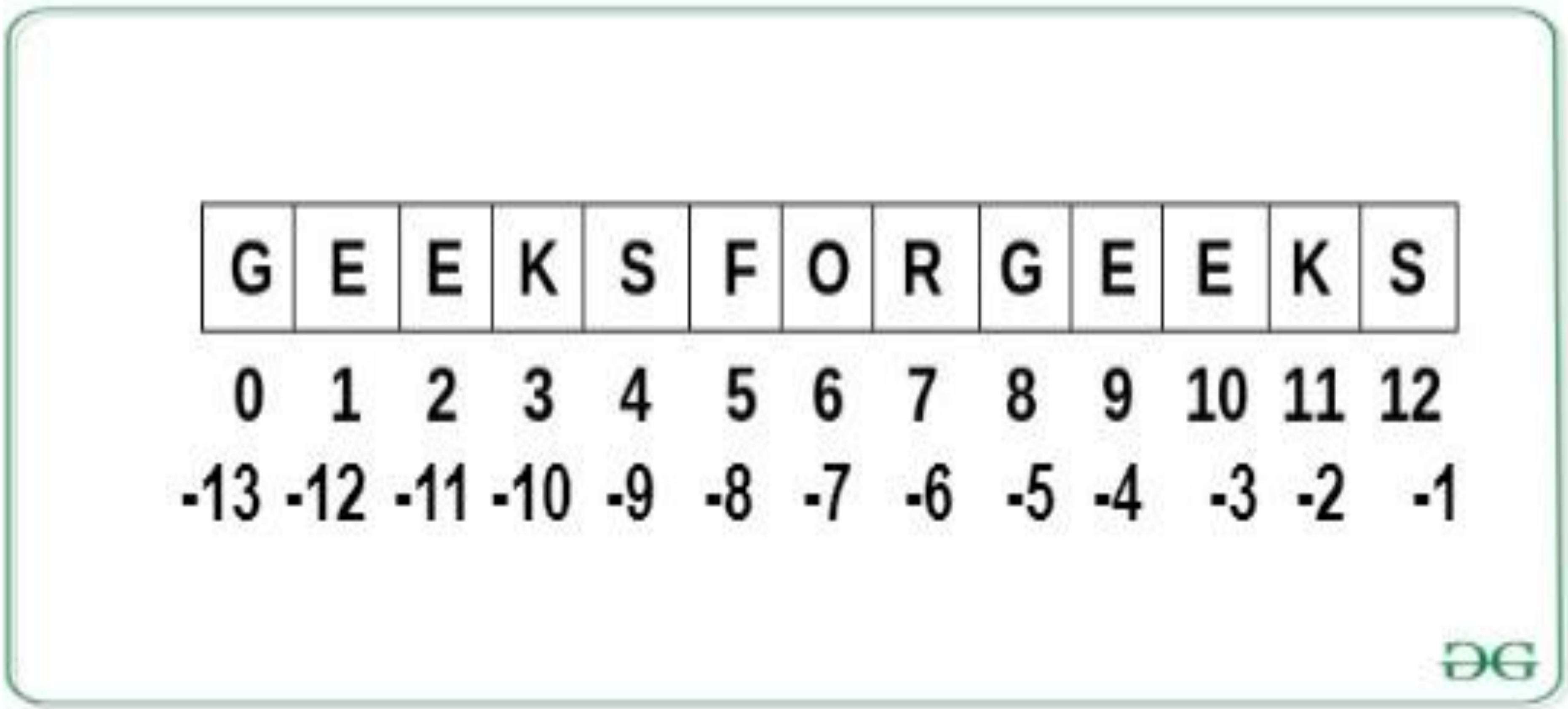
- Python does not have a character data type, a single character is simply a string with a length of 1.

❖**Note:** As strings are immutable, modifying a string will result in creating a new copy.

Accessing characters in Python String

- In Python Programming tutorials, individual characters of a String can be accessed by using the method of Indexing.
- Python strings are zero-indexed, meaning the first character is at index 0, the second character at index 1, and so on.
- Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character, and so on.
- While accessing an index out of the range will cause an IndexError. Only Integers are allowed to be passed as an index, float or other types that will cause a TypeError.

Python String Indexing



Example: Python Strings Operations

```
String = "Welcome to GeeksForGeeks"
print("Creating String: ")
print(String)
```

✓*Output*

```
Creating String:
Welcome to GeeksForGeeks
```

```
# Printing First character
print("\nFirst character of String is: ")
print(String[0])

# Printing Last character
print("\nLast character of String is: ")
print(String[-1])
```

✓*Output*

```
First character of String is:
W
Last character of String is:
s
```