PNS SCHOOL OF ENGINEERING & TECHNOLOGY

Nishamani Vihar, Marshaghai, Kendrapara

LECTURE NOTES ON PYTHON PROGRAMMING

3rd Semester

PREPARED BY

Miss. Sushree Sangita Tripathy

LECTURER IN COMPUTER SCIENCE & ENGINEERING

1.0 Introduction to Python

1.1 Overview of Python: Features and Applications, Setting Up the Environment

Python is a high-level, interpreted, and general-purpose programming language that emphasizes code readability and simplicity. Created by Guido van Rossum and first released in 1991, it has become one of the most popular programming languages in the world.

Features of Python:

- Simple and easy to learn
- Interpreted language
- Dynamically typed
- Extensive standard libraries
- Open-source and community-driven
- Portable and cross-platform
- Supports multiple programming paradigms (procedural, object-oriented, functional)

Applications of Python:

- Web Development (e.g., Django, Flask)
- Data Science and Machine Learning (e.g., Pandas, NumPy, Scikit-learn)
- Scripting and Automation
- Game Development
- Desktop GUI Applications
- Network Programming
- IoT and Embedded Systems
- Cybersecurity and Penetration Testing

Setting Up the Python Environment:

To start programming in Python, you need to install Python and choose an Integrated Development Environment (IDE).

1. Download and install Python from the official website:
https://www.python.org/downloads/

2. Choose an IDE or code editor:

- IDLE (default IDE with Python)
- VS Code
- PyCharm
- Jupyter Notebook (for data science)

1.2 Python Syntax: Variables, Data Types, and Operators

Variables:
Variables are used to store data. In Python, you do not need to declare the type of a variable explicitly.

Example:

```
x = 10
name = 'Alice'
pi = 3.14
```

Data Types:
- int: Integer numbers
- float: Floating-point numbers
- str: String (text)
- bool: Boolean (True/False)
- list, tuple, set, dict: Collection types

Operators:
- Arithmetic Operators: +, -, *, /, %, //, **
- Comparison Operators: ==, !=, >, <, >=, <=
- Logical Operators: and, or, not
- Assignment Operators: =, +=, -=, *=, /=, etc.

Writing, Executing, and Debugging Python Scripts
Python scripts can be written in any text editor and saved with a `.py` extension. They can be executed using the command line or an IDE.

To run a script from the terminal:

```
python filename.py
```

Debugging:
- Use print() statements to trace values.
- Use IDE built-in debugging tools.
- Use the `pdb` module for step-by-step debugging.

2.0 Control Structures and Functions

2.1 Conditional Statements and Loops

Conditional Statements: if, else, elif
Conditional statements are used to perform different actions based on different conditions.
- `if`: Executes a block of code if the condition is true.
- `elif`: Executes a block of code if the previous condition(s) are false and this one is true.
- `else`: Executes a block of code if all previous conditions are false.

Example:

```
x = 10
if x > 0:
    print("Positive")
elif x == 0:
    print("Zero")
else:
    print("Negative")
```

Loops: for, while, and Nested Loops

Loops are used to execute a block of code repeatedly.
- `for` loop: Iterates over a sequence (like list, tuple, string).
- `while` loop: Repeats as long as a condition is true.
- Nested loops: A loop inside another loop.

Example:

```
# for loop
for i in range(5):
    print(i)

# while loop
count = 0
while count < 5:
    print(count)
    count += 1

# Nested loop
for i in range(3):
    for j in range(2):
        print(f"i={i}, j={j}")
```

2.2 Functions

Defining, Calling, and Scope of Variables

Functions are blocks of code that perform a specific task.
- Define a function using `def` keyword.
- Call the function by its name followed by parentheses.
- Variables inside a function are local unless declared global.

Example:

```
def greet(name):
    message = f"Hello, {name}!"
    return message
```

```
print(greet("Alice"))
```

Introduction to Lambda Functions and Recursion

Lambda functions are anonymous functions defined with the `lambda` keyword. They are often used for short, throwaway functions.

Example:

```
square = lambda x: x * x
print(square(4))
```

Recursion is a technique where a function calls itself to solve a problem.

Example:

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n - 1)

print(factorial(5))
```

3.0 Data Structures in Python

3.1 Lists, Tuples, Sets, and Dictionaries: Operations and Applications

Python provides several built-in data structures that are versatile and easy to use:

Lists:

Lists are ordered, mutable collections.
Example:

```
fruits = ['apple', 'banana', 'cherry']
fruits.append('orange')
print(fruits[1])
```

Tuples:

Tuples are ordered and immutable collections.
Example:

```
coordinates = (10, 20)
print(coordinates[0])
```

Sets:

Sets are unordered collections of unique items.
Example:

```
unique_numbers = {1, 2, 3, 4}
```

```
unique_numbers.add(5)
print(unique_numbers)
```

Dictionaries:

Dictionaries store key-value pairs.
Example:

```
student = {'name': 'Alice', 'age': 21}
print(student['name'])
```

List Comprehensions:

List comprehensions provide a concise way to create lists.
Example:

```
squares = [x*x for x in range(5)]
```

Dictionary Comprehensions:

Similar to list comprehensions but used for dictionaries.
Example:

```
square_dict = {x: x*x for x in range(5)}
```

3.2 Working with Strings: Methods and Manipulation

Strings in Python are immutable sequences of characters. Common string methods include:
- `upper()`, `lower()`
- `strip()`, `replace()`
- `split()`, `join()`
- `find()`, `count()`

Example:

```
text = '  Hello World!  '
print(text.strip().lower())
```

3.3 Introduction to Python's Collections Module

The `collections` module provides specialized container datatypes:
- `namedtuple()`: Tuple with named fields.
- `deque`: List optimized for fast appends and pops.
- `Counter`: Dict subclass for counting hashable objects.
- `OrderedDict`: Dict that remembers insertion order.
- `defaultdict`: Dict with a default value for missing keys.

Example:

```
from collections import Counter
counts = Counter(['apple', 'banana', 'apple'])
print(counts)
```