Module 13: Python Fundamentals

1. Introduction to Python

Features of Python

- Simple, easy-to-read syntax
- Object-oriented
- Open-source, large community
- Portable (runs on multiple platforms)
- Extensive standard library
- Dynamically typed

History and Evolution

- Developed in late 1980s by **Guido van Rossum** in the Netherlands (CWI)
- First released publicly in **1991**
- Major releases: Python 2 (2000), Python 3 (2008)
- Widely used in web development, data science, automation, artificial intelligence, etc.

Advantages

- Fast development time due to readable syntax
- Extensive library ecosystem
- Versatile application (web, scripting, data, automation)
- Robust community support

Installing Python

- Download from python.org
- Use installers or `pip`
- Set up with any IDE (VS Code, PyCharm, Anaconda, etc.)

Writing and Running First Program

Lab Example:

```
print("Hello, World!")
```

2. Programming Style (PEP 8)

Key Points

- **Indentation:** 4 spaces per block
- **Comments:** Use `#` for single-line comments
- **Naming:** Descriptive variable and function names in snake_case

Lab Example:

```
def sum1(a, b):
    return a + b

n1 = 10
n2 = 20
result = sum1(n1, n2)
print("Sum:", result)
```

3. Core Python Concepts

Common Data Types

- **int:** 10
- **float:** 3.14
- **string:** "hello"
- **list:**[1, "apple", 3.14]
- **tuple:** (1, 2, 3)
- **dictionary:** {"name": "A"}
- **set:** {1, 2, 3}

Lab Example:

```
an_int = 10
a_float = 5.62
a_str = "Python"
a_list = [1, 2, 3]
a_tuple = (4, 5, 6)
a_dict = {"key": "value"}
a_set = {7, 8, 9}
print(type(a_float))   # Output: <class 'float'>
```

4. Conditional Statements

Theory

- **if, elif, else** used for decision making
- **Nested if**: An if inside another if block

Lab Examples:

(a) Check Positive, Negative, or Zero

```
num = int(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num < 0:
    print("Negative number")
else:
    print("Zero")
```

(b) Prime Number Check

```
num = int(input("Enter a number: "))
is_prime = True
if num <= 1:
    is_prime = False
else:
    for i in range(2, num):
```

```
        if num % i == 0:
            is_prime = False
            break
if is_prime:
    print(num, "is Prime")
else:
    print(num, "is Not Prime")
```

(c) Grade Calculation

```
percentage = float(input("Enter percentage: "))
if percentage >= 90:
    print("Grade: A")
elif percentage >= 80:
    print("Grade: B")
elif percentage >= 60:
    print("Grade: C")
else:
    print("Grade: D")
```

(d) Blood Donation Eligibility

```
age = int(input("Enter age: "))
weight = float(input("Enter weight in kg: "))
if age >= 18:
    if weight >= 50:
        print("Eligible to donate blood")
    else:
        print("Not eligible: Weight less than 50kg")
else:
    print("Not eligible: Age below 18")
```

5. Looping (For, While)

Theory

- **For loop:** Iterate over sequences (lists, strings, etc.)
- **While loop:** Repeat as long as condition is true

Lab Examples:

(a) Print Fruits in a List

```
List1 = ['apple', 'banana', 'mango']
for fruit in List1:
    print(fruit)
```

(b) Print Length of Each String

```
for fruit in List1:
    print(fruit, "length:", len(fruit))
```

(c) Search for Specific String

```
search_item = "banana"
found = False
for fruit in List1:
    if fruit == search_item:
        found = True
        print(search_item, "is in the list")
if not found:
    print(search_item, "not found in the list")
```

(d) Print Pattern with Nested Loop

```
for i in range(1, 6):
    print("*" * i)
```

6. Generators and Iterators

Theory

- **Generators:** Functions that yield values one at a time using `yield`
- **Iterators:** Objects implementing `__iter__()` and `__next__()`

Lab Examples:

(a) Generator for First 10 Even Numbers

```
def even_numbers():
    for i in range(1, 21):
        if i % 2 == 0:
            yield i

for even in even_numbers():
    print(even, end=' ')
print()
```

(b) Custom Iterator

```
class ListIterator:
    def __init__(self, items):
        self.items = items
        self.index = 0
    def __iter__(self):
        return self
    def __next__(self):
        if self.index < len(self.items):
            item = self.items[self.index]
            self.index += 1
            return item
        raise StopIteration()

numbers = [1, 2, 3, 4]
for n in ListIterator(numbers):
    print(n)
```

7. Functions and Methods

Theory

- Functions are defined using `def`
- Arguments can be positional, keyword, or have default values
- Variables in functions are local unless `global` is used

Lab Examples:

```
# 1. Print "Hello"
print("Hello")

# 2. Assign string to a variable, print
msg = "Welcome"
print(msg)

# 3. Triple quote string
print("""This is a string using triple quotes""")

# 4. Indexing strings
s = "Python"
print(s[0])      # First character
print(s[1:])     # Second to end
print(s[:5])     # Up to 5th
print(s[1:5])    # Between 1 and 4
print(s[-1])     # Last character
print(s[1::2])   # Every alternate character from 1
```

8. Control Statements (Break, Continue, Pass)

Theory:

- `break`: Exits the loop completely
- `continue`: Skips the rest of the current iteration
- `pass`: Does nothing

Lab Examples:

```
List1 = ['apple', 'banana', 'mango']

# Skip 'banana' with continue
for fruit in List1:
    if fruit == 'banana':
        continue
    print(fruit)

# Stop loop once 'banana' found with break
for fruit in List1:
```

```
    if fruit == 'banana':
        break
    print(fruit)
```

9. String Manipulation

Theory:

- Concatenation: `"a" + "b"`
- Repetition: `"a" * 3`
- Useful methods: `upper()`, `lower()`, `strip()`, `split()`, `replace()`
- Slicing: `string[start:end]`

Lab Examples:

```
s = "hello world"
print(s[0:5])         # 'hello'
print(s[6:])          # 'world'
print(s.upper())      # 'HELLO WORLD'
print(s.capitalize()) # 'Hello world'
print(s.replace("world", "Python"))
```

10. Advanced Python: map, reduce, filter, Closures, Decorators

Theory:

- **map()**: Apply function to every element
- **filter()**: Select elements that pass a test
- **reduce()**: Cumulatively apply a function (must `import` from `functools`)
- **Closure**: Inner function remembers environment of enclosing function
- **Decorator**: Function that adds extra capability to another function

Lab Examples:

```
# Using map() to square numbers
nums = [1, 2, 3, 4]
squared = list(map(lambda x: x*x, nums))
```

```
print(squared)

# Using reduce() to multiply numbers
from functools import reduce
product = reduce(lambda x, y: x * y, nums)
print(product)

# Using filter() to get even numbers
evens = list(filter(lambda x: x % 2 == 0, nums))
print(evens)
```

Assessment: Mini Project

Simple Grade Management System

```
def calculate_grade(percentage):
    if percentage >= 90:
        return 'A'
    elif percentage >= 80:
        return 'B'
    elif percentage >= 60:
        return 'C'
    else:
        return 'D'

def main1():
    n = int(input("Enter number of students: "))
    for i in range(n):
        name = input("Enter student name: ")
        percentage = float(input("Enter percentage: "))
        grade = calculate_grade(percenatge)
        print(f"Student: {name}, Percentage: {percentage}, Grade: {grade}")

if __name__ == "__main__":
    main1()
```