PYTHON
PROGRAMMING
STUDY NOTES

Computer Science | External Examination Preparation

Covers: Variables • Data Types • Operators • Lists • Conditions • Loops • Functions

01 Python Introduction

Python is a high-level, interpreted, general-purpose programming language. It was created by Guido van Rossum and first released in 1991. Python emphasizes readability with clean, simple syntax. It is widely used in web development, data science, artificial intelligence, automation, and more.

Python code runs line by line using an interpreter. No compilation step is needed — you write code and it runs immediately.

Code Example:

```python
# This is our very first Python program
print("Hello, World!")
print("Welcome to Python Programming")

# Python can also do math directly
print(2 + 3)
print(10 * 5)
```

Output:

```
Hello, World!
Welcome to Python Programming
5
50
```

02 Python Comments

Comments are lines in your code that Python ignores during execution. They are used to explain the code, making it easier for you and others to understand. Comments are an essential part of writing clean, professional code.

Single-line comments start with a # symbol. Multi-line comments use triple quotes """ ... """.

Code Example:

```python
# This is a single-line comment
print('Hello')   # Comments can also appear after code

 # This is a multiple-line comment
"""
This is a multi-line comment.
It can span across many lines.
Python will ignore all of this.
"""
print('Comments do not affect output')
```

Output:

```
Hello
Comments do not affect output
```

03 Print in Python (Basic Input / Output)

The print() function is used to display output on the screen. It is one of the most commonly used functions in Python. The input() function is used to receive data from the user through the keyboard.

Code Example:

```
# Basic print
print("My name is Ali")
print("I am", 15, "years old")

# Displays output
print("Apple", "Banana", "Mango")
print("Hello World")

# Takes input from the user
name = input("Enter your name: ")
print("Hello,", name)
```

Output:

```
My name is Ali
I am 15 years old
Apple Banana Mango
Hello World
Enter your name: Ahmed
Hello, Ahmed
```

04 Python Variables

A variable is a container that stores data values in memory. In Python, you do not need to declare the type of a variable — Python figures it out automatically. Variable names are case-sensitive and should be meaningful.

Rules:

- **Must start with a letter or an underscore:** (age, _name are valid; 1name is invalid).
- **Can only contain alphanumeric characters and underscores:** (A-z, 0-9, and _). Special characters like spaces, dashes (-), or $ are not allowed.
- **Cannot be a Python keyword or reserved word:** Words like if, for, class, print, break, else, None, True, and False have special meanings and cannot be used as variable names.
- **Are case-sensitive:** age, Age, and AGE are considered three different variables.

Code Example:

```
# Creating variables
student_name = 'Sara'
age = 16
gpa = 3.85
is_enrolled = True

```

```
# Using variables
print(student_name)
print('Age:', age)
print('GPA:', gpa)

```

Output:

```
Sara
Age: 16
GPA: 3.85

```

05 Casting in Python (Type Conversion)

Casting means converting a variable from one data type to another. This is useful when you receive user input (which is always a string) and need to perform calculations on it. Python provides built-in functions: int(), float(), str(), bool().

Code Example:

```
# Converting between types
num_str = "42"
num_int = int(num_str)     # String → Integer
num_float = float(num_str) # String → Float

print(num_int + 8)         # 50
print(num_float)           # 42.0
print(str(100) + ' kg')    # Integer → String

# Practical example: user input
age = int(input("Enter age: "))
print('Next year you will be', age + 1)
```

Output:

```
50
42.0
100 kg
Enter age: 17
Next year you will be 18
```

06 Numeric Datatypes (int & float)

Python has two main numeric types: int (whole numbers like 5, -10, 1000) and float (decimal numbers like 3.14, -0.5, 2.0). Python automatically handles very large integers without any limits.

Code Example:

```
# Integer examples
x = 10
y = -25
big = 1_000_000    # Underscore for readability
print(type(x))     # <class 'int'>

# Float examples
pi = 3.14159
temp = -5.5
print(type(pi))    # <class 'float'>

# Math operations
print(10 // 3)     # Floor division = 3
print(10 % 3)      # Modulus (remainder) = 1
print(2 ** 8)      # Power = 256
```

Output:

```
<class 'int'>
<class 'float'>
3
1
256
```

07 String Datatype

Strings are sequences of characters enclosed in single quotes ('...') or double quotes ("..."). Strings are immutable — once created, they cannot be changed directly. Python provides powerful built-in methods for working with strings.

Code Example:

```
word = 'Python Programming'

# Slicing strings
print(word[0:6])        # Python
print(word[-11:])       # Programming
print(word[::2])        # Every 2nd character

# String methods
print(word.upper())     # PYTHON PROGRAMMING
print(word.lower())     # python programming
print(word.replace('Python', 'Java'))
print(len(word))        # 18


```

```
# Concatenation & formatting
name = 'Ali'
age = 17
print('Name: ' + name)
print(f"My name is {name} and I am {age} years old.")
```

Output:

```
Python
Programming
Pto rgamn
PYTHON PROGRAMMING
python programming
Java Programming
18
Name: Ali
My name is Ali and I am 17 years old.
```

08 Boolean Datatype (True / False)

Boolean is a data type that has only two possible values: True or False. Booleans are the result of comparison and logical operations. They are essential for controlling the flow of a program through conditions and loops.

Code Example:

```
# Boolean values
is_sunny = True
is_raining = False

print(type(is_sunny))     # <class 'bool'>
print(10 > 5)             # True
print(10 == 5)            # False

# bool() conversion
print(bool(0))            # False
print(bool(1))            # True
print(bool(''))           # False (empty string)
print(bool('Hello'))      # True (non-empty)
```

Output:

```
<class 'bool'>
True
False
False
True
False
True
```

09 Operators

Operators are symbols that perform operations on variables and values. Python has several categories of operators:

Code Example:

```
# Arithmetic Operators
print(10 + 3)    # Addition      = 13
print(10 - 3)    # Subtraction   = 7
print(10 * 3)    # Multiply      = 30
print(10 / 3)    # Division      = 3.333...
print(10 // 3)   # Floor div     = 3
print(10 % 3)    # Modulus       = 1
print(2 ** 10)   # Power         = 1024

# Comparison Operators
x = 10
print(x == 10)   # True    (equal)
print(x != 5)    # True    (not equal)
print(x > 8)     # True
print(x <= 10)   # True

# Logical Operators
print(True and False)   # False
print(True or False)    # True
print(not True)         # False

# Assignment Operators
n = 5
n += 3   # Same as n = n + 3
print(n) # 8
```

Output:

```
13
7
30
3.3333333333333335
3
1
1024
True
True
True
True
False
True
False
```

```
8
```

10 List / Array Datatype

A list is an ordered, changeable collection that allows duplicate values. Lists are one of the most versatile data structures in Python. Items are accessed by index starting at 0.

Code Example:

```
# Creating a list
fruits = ['Apple', 'Banana', 'Mango', 'Orange']

# Access items
print(fruits[0])        # Apple
print(fruits[-1])       # Orange (last item)

# Change items
fruits[1] = 'Grapes'
print(fruits)

# Add items
fruits.append('Kiwi')           # Add to end
fruits.insert(1, 'Strawberry') # Add at index 1

# Remove items
fruits.remove('Apple')          # By value
fruits.pop(0)                   # By index

print(len(fruits))      # Length of list
print(fruits)
```

Output:

```
Apple
Orange
['Apple', 'Grapes', 'Mango', 'Orange']
4
['Mango', 'Orange', 'Kiwi']
```

11 Tuples, Sets & Dictionaries

Tuple: An ordered, unchangeable (immutable) collection. Once created, items cannot be changed. Good for fixed data like coordinates or days of the week.

Set: An unordered collection with no duplicate values. Useful for storing unique items and performing set operations (union, intersection).

Dictionary: A collection of key-value pairs. Each key maps to a value, like a word and its definition.

Code Example:

```
# Tuple
days = ('Mon', 'Tue', 'Wed', 'Thu', 'Fri')
print(days[0])            # Mon
print(len(days))          # 5

# Set (no duplicates)
nums = {1, 2, 3, 2, 1}
print(nums)               # {1, 2, 3}

# Dictionary
student = {
  'name': 'Ahmed',
  'age': 16,
  'grade': 'A'
}
print(student['name'])    # Ahmed
student['age'] = 17       # Update value
student['city'] = 'Islamabad'   # New key
print(student)
```

Output:

```
Mon
5
{1, 2, 3}
Ahmed
{'name': 'Ahmed', 'age': 17, 'grade': 'A', 'city': 'Islamabad'}
```

12 Conditions (if / elif / else)

Conditions allow a program to make decisions and execute different code based on whether a condition is True or False. Python uses indentation (4 spaces) to define code blocks.

Code Example:

```
marks = 78

# if condition
if marks >= 90:
    print('Grade: A+')
elif marks >= 80:
    print('Grade: A')
elif marks >= 70:
    print('Grade: B')
elif marks >= 60:
    print('Grade: C')
```

```
else:
    print('Grade: F - Please study harder!')

# Nested condition
age = 18
has_id = True
if age >= 18:
    if has_id:
        print('Access granted')
    else:
        print('ID required')
```

Output:

```
Grade: B
Access granted
```

13 Match - Case Statements

Match-case is Python's version of a switch statement (added in Python 3.10). It compares a value against multiple patterns and executes the matching block. It is cleaner and more readable than many if-elif chains.

Code Example:

```
day = 'Monday'

match day:
    case 'Monday':
        print('Start of the work week!')
    case 'Friday':
        print('Almost the weekend!')
    case 'Saturday' | 'Sunday':
        print('It is the weekend!')
    case _:
        print('A regular weekday')

# Match with numbers
score = 9
match score:
    case 9:
        print('Excellent!')
    case 8:
        print('Good job!')
    case _:
        print('Keep trying!')
```

Output:

```
Start of the work week!
Excellent!
```

14 While Loop

A while loop repeatedly executes a block of code as long as a condition remains True. It is used when you do not know in advance how many times the loop should run. Be careful to avoid infinite loops — always update the condition variable inside the loop.

Code Example:

```python
# Basic while loop
count = 1
while count <= 5:
    print('Count:', count)
    count += 1


# Guess the number game
secret = 7
guess = int(input('Guess the number: '))
while guess != secret:
    print('Wrong! Try again.')
    guess = int(input('Guess: '))
print('Correct! You guessed it!')
```

Output:

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
Guess the number: 5
Wrong! Try again.
Guess: 7
Correct! You guessed it!
```

15 Functions

A function is a reusable block of code that performs a specific task. Functions help organize code, avoid repetition, and make programs easier to read and maintain. You define a function with the def keyword, give it a name, and optionally pass parameters to it.

Code Example:

```
def add(a, b):
    Return a + b

print(add(50, 20)    # Calling the function

def subtract(a, b):
    return a - b

result = subtract(25, 10)      # storing the values in variables
print('Sum:', result)

def farenheit_to_celsius(fahrenheit):
    return (fahrenheit - 32) * 5/9

print(farenheit_to_celsius(100))
print(farenheit_to_celsius(96))
```

Output:

```
70
Subtract: 15
37.77777777777778
35.55555555555556
```