Python Programming Reviewer – Basic Concepts

Python Creator
Python was created by **Guido van Rossum** in the late 1980s and officially released in 1991.
Example: Guido developed Python to be simple, readable, and powerful for everyday tasks.

Language Classification
Python is a **high-level interpreted language**, meaning it executes line by line and focuses on readability.
Example: Unlike C (compiled), Python doesn't need a separate compile step before running.

File Extension
Python source files use the **".py"** extension.
Example: hello.py contains Python code that can be run with python hello.py.

Displaying Output
The **print()** function displays text or variables on the screen.
Example: print("Hello, World!") outputs → Hello, World!

Comments in Python
Comments start with the **#** symbol. Python ignores these lines.
Example:

```
# This is a comment

print("Hi")  # Inline comment
```

Valid Variable Names
Variables can't start with numbers or include spaces/dashes.

Valid Example: my_name = "Buddy"

❌ Invalid: 2name, my-name, my name

Defining a Function

Functions are defined using the keyword **def**.

Example:

```
def greet():
  print("Hello!")
```

Arithmetic Operation

print(3 + 4) outputs **7** because + adds numbers.

Example: print(5 + 10) → 15

Taking User Input

Use **input()** to read user input as a string.

Example:

```
name = input("Enter your name: ")
print("Hello", name)
```

Data Types in Python

Basic types include **int**, **float**, **str**, and **bool** — not "real".

Example:

```
x = 5     # int
y = "Hi"   # str
z = True   # bool
```

Variable Type Example

x = "Hello World" assigns a **string** (str) value.

Python determines types automatically — no need for declaration.

Exponentiation Operator

Use ****** for powers.

Example: 2 ** 3 = 8

Comparison Operator

== checks **equality**, not assignment.

Example:

```
x = 5

print(x == 5)  # True
```

Conditional Statement

if is used for **conditional logic**.

Example:

```
if x > 0:
   print("Positive")
```

Looping Keywords

Loops in Python begin with **for** or **while**.

Example:

```
for i in range(3):
   print(i)
```

Break Statement

break exits a loop early.

Example:

```
for i in range(5):
   if i == 3:
      break
```

Logical Operators

Python uses **and**, **or**, **not** — not “then”.

Example:

```
if x > 0 and x < 10:
   print("Single digit")
```

Type Conversion

Convert "25" to integer using **int("25")**.

Example:

```
age = int("25")
print(age + 1)  # 26
```

Length Function

len() returns the length of a string, list, or tuple.

Example:

```
len("Python")  # 6
```

String Multiplication

"Python" * 2 repeats the string → **PythonPython**

Example:

```
print("Hi" * 3)  # HiHiHi
```

Python Programming Applied Concepts

Defining Functions

Functions in Python are created using the **def** keyword followed by the function name and parentheses.

Example:

```
def greet():
```

```
print("Hello!")
```

Function Output Example

```
def add(x, y):
    return x + y

print(add(2, 3))
```

✅ Output: **5**
The return statement sends back the result of x + y.

Purpose of return
return is used to **send a value back** to the function's caller — it doesn't print directly.
Example:

```
def square(x):
    return x * x
```

Lists in Python
A **list** is defined using square brackets [ ].
Example:

```
fruits = ["apple", "banana", "cherry"]
```

Mutability of Lists
Lists are **mutable**, meaning they can be changed (add, remove, or modify items).
Example:

```
fruits[1] = "mango"
```

List Indexing
Indexing starts at **0**.

```
fruits = ["apple", "banana", "cherry"]
```

```
print(fruits[1])  # banana
```

Adding Elements to a List

Use **append()** to add an item at the end of a list.

Example:

```
fruits.append("orange")
```

Removing Items from a List

Use **remove()** to delete a specific item.

Example:

```
fruits.remove("banana")
```

Lists vs. Tuples

- **Lists** → Mutable (can change)
- **Tuples** → Immutable (cannot change)
  Example:

```
my_list = [1, 2, 3]

my_tuple = (1, 2, 3)
```

Dictionaries

Dictionaries store **key–value pairs** using curly braces {}.

Example:

```
person = {"name": "John", "age": 25}
```

Dictionary Keys

Keys must be **unique and immutable** (e.g., strings, numbers).

Example:

```
{"id": 101, "name": "Anna"}
```

Accessing Dictionary Values

Access values using the **key name in brackets**.

Example:

```
student = {"name": "Anna", "age": 18}

print(student["age"])  # 18
```

Sets in Python

Sets are defined with { } and automatically remove duplicates.

Example:

```
nums = {1, 2, 3, 3}

print(nums)  # {1, 2, 3}
```

Set Property

Sets automatically **remove duplicates** and are unordered.

The in Operator

Checks **membership** in a list, set, or string.

Example:

```
if "apple" in fruits:
   print("Found!")
```

String Methods

.lower() converts a string to lowercase.

Example:

```
text = "Python"

print(text.lower())  # python
```

Splitting Strings

Use **split()** to divide a string into a list of words.

Example:

```
message = "Hello World"

print(message.split())  # ['Hello', 'World']
```

Opening Files

To open a file for reading, use **open("filename", "r")**.

Example:

```
file = open("data.txt", "r")
```

Safe File Handling

Use **with open(...) as** to automatically close files after use.

Example:

```
with open("data.txt", "r") as f:
    data = f.read()
```

Printing with Variables

You can print variables alongside text using commas in print().

Example:

```
name = "Alice"

print("Hello", name)  # Hello Alice
```

Logic, Files & OOP

Loop Execution Count

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

The loop runs **5 times** (0 to 4).

✅ Output: 0 1 2 3 4

The range(3) Sequence

range(3) produces **0, 1, 2** — it starts at 0 and stops *before* 3.

While Loop Example

```
x = 0
while x < 3:
  print("Hi")
  x += 1
```

✅ Output: “Hi” three times

Range with Start and End

```
for n in range(2, 6):
  print(n)
```

✅ Output: 2 3 4 5

The upper limit (6) is *not included*.

Skipping Iterations

The **continue** statement skips the current iteration and moves to the next.

Example:

```
for i in range(5):
  if i == 2:
    continue
  print(i)
```

Comparison in Conditions

if score >= 75: means **score is greater than or equal to 75**.

Even or Odd Check

```
num = 10
if num % 2 == 0:
  print("Even")
```

✅ Output: **Even**

Creating a Class

A class is created using the keyword **class**.

Example:

```
class Car:
  pass
```

The __init__() Method

__init__() is a **constructor** — it initializes object attributes when the object is created.

Example:

```
class Dog:
  def __init__(self, name):
    self.name = name
```

Class and Method Example

```
class Dog:
  def bark(self):
    print("Woof!")
d = Dog()
d.bark()
```

✅ Output: **Woof!**

Understanding Objects

Objects are **instances of classes**, meaning they are created from class blueprints.

Creating an Object

You create an object by **calling the class name with parentheses**.

Example:

```
car = Car()
```

Exception Handling Example

```
try:
  print(10 / 0)
except ZeroDivisionError:
  print("Cannot divide by zero!")
```

✅ Output: **Cannot divide by zero!**

Handling Exceptions

The keyword **except** is used to handle errors gracefully in Python.

Writing to Files

```
with open("test.txt", "w") as f:
  f.write("Hello")
print("Done")
```

✅ Writes “Hello” to the file and prints **Done**.

File Modes

'r+' mode allows both **reading and writing** to a file.

Input Type

The **input()** function always returns a **string**, even if you type numbers.

Example:

```
age = input("Enter age: ")

print(type(age))  # str
```

Loop with List Multiplication

```
for x in [1, 2, 3]:
   print(x * 2)
```

✅ Output: **2 4 6**

---

len() Function

len() works on **strings, lists, and tuples** to return their length.

Example:

```
len([1, 2, 3])  # 3

len("Python")   # 6
```

Modulus Operator

10 % 3 gives the **remainder** of division → **1**.

Example:

```
print(10 % 3)  # 1
```