PGDCA SEM – 2
CORE PYTHON PROGRAMMING

Q-1(A) Answer the following: **04**

1. **Define: Memory Management.**
   Answer: Memory management in Python involves the management of a private heap.
2. **Define: Garbage Collection.**
   Answer: A garbage collection in Python manages the memory automatically and heap allocation.
3. **Define: Sets.**
   Answer: Sets are used to store multiple items in a single variable.
4. **Define: Identifiers.**
   Answer: Python Identifier is the name we give to identify a variable, function, class, module or other object.

Q-1(B) Attempt Any One (Out of Two): **02**

1. **What are Literals?**
   Answer: Python literals are a data type and can hold any value type, such as strings, numbers, and more. Because Python literals are used to store base data for the source code of your software, they need to be robust enough to work with any data. Python literals can hold any data you need; let's look at some examples next.
2. **What is Naming Conventions?**
   Answer: Naming conventions in Python refer to a set of rules and guidelines for naming variables, functions, classes, and other entities in your code. Adhering to these conventions ensures consistency, readability, and better collaboration among developers.

Q-1(C) Attempt Any One (Out of Two): **03**

1. **Explain: Features Of Python**
   Answer: 1. Free and Open Source.
   2. Easy to code.
   3. Easy to read.
   4. Object-Oriented Language.
   5. GUI Programming Support.
   6. High Level Language.
   7. Large Community Support.
   8. Easy to Debug.

2. **Explain: Built In Datatypes in Python.**
   Answer: Python has several built-in data types, including numeric types (int, float, complex), string (str), boolean (bool), and collection types (list, tuple, dict, set). Each data type has its own set of properties, methods, and behaviors that allow programmers to manipulate and process data effectively in their programs.

Q-1(D) Attempt Any One (Out of Two): **05**

1. **Explain Operators in Python.**
   Answer: In Python programming, Operators in general are used to perform operations on values and variables. These are standard symbols used for logical and arithmetic operations. In this article, we will look into different types of Python operators.
   Types of Operators in Python
   A. Arithmetic Operators
   B. Comparison Operators
   C. Logical Operators
   D. Bitwise Operators

   A. Arithmetic Operators in Python
   Python Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication, and division.
   In Python 3.x the result of division is a floating-point while in Python 2.x division of 2 integers was an integer. To obtain an integer result in Python 3.x floored (// integer) is used.

+	Addition: adds two operands	x + y
–	Subtraction: subtracts two operands	x – y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y

%	Modulus: returns the remainder when the first operand is divided by the second	x % y
**	Power: Returns first raised to power second	x ** y

B. Comparison of Python Operators

In Python Comparison of Relational operators compares the values. It either returns True or False according to the condition.

>	Greater than: True if the left operand is greater than the right	x > y
<	Less than: True if the left operand is less than the right	x < y
==	Equal to: True if both operands are equal	x == y
!=	Not equal to – True if operands are not equal	x != y
>=	Greater than or equal to True if the left operand is greater than or equal to the right	x >= y
<=	Less than or equal to True if the left operand is less than or equal to the right	x <= y

C. Logical Operators in Python

Python Logical operators perform Logical AND, Logical OR, and Logical NOT operations. It is used to combine conditional statements.

and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

D. Bitwise Operators in Python

Python Bitwise operators act on bits and perform bit-by-bit operations. These are used to operate on binary numbers.

&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

2. **Write Steps to install and configure Python.**

Answer: The installation requires downloading the official Python .exe installer and running it on your system. The sections below will explain several options and details during the installation process.

Step 1: Select Python Version

Deciding on a version depends on what you want to do in Python. The two major versions are Python 2 and Python 3. Choosing one over the other might be better depending on your project details. If there are no constraints, choose whichever one you prefer.

We recommend Python 3, as Python 2 reached its end of life in 2020. Download Python 2 only if you work with legacy scripts and older projects. Also, choose a stable release over the newest since the newest release may have bugs and issues.

Step 2: Download Python Executable Installer

Start by downloading the Python executable installer for Windows:

1. Open a web browser and navigate to the Downloads for Windows section of the official Python website.
2. Locate the desired Python version.
3. Click the link to download the file. Choose either the Windows 32-bit or 64-bit installer.

The download is approximately 25MB.

Step 3: Run Executable Installer

The steps below guide you through the installation process:

1. Run the downloaded Python Installer.
2. The installation window shows two checkboxes:

Admin privileges. The parameter controls whether to install Python for the current or all system users. This option allows you to change the installation folder for Python.

Add Python to PATH. The second option places the executable in the PATH variable after installation. You can also add Python to the PATH environment variable manually later.

For the most straightforward installation, we recommend ticking both checkboxes.

3. Select the Install Now option for the recommended installation (in that case, skip the next two steps).

To adjust the default installation options, choose Customize installation instead and proceed to the following step.

The default installation installs Python to C:\Users\[user]\AppData\Local\Programs\Python\Python[version] for the current user. It includes IDLE (the default Python editor), the PIP package manager, and additional documentation. The installer also creates necessary shortcuts and file associations.

Customizing the installation allows changing these installation options and parameters.

4. Choose the optional installation features. Python works without these features, but adding them improves the program's usability.
Click Next to proceed to the Advanced Options screen.
5. The second part of customizing the installation includes advanced options.
Choose whether to install Python for all users. The option changes the install location to C:\Program Files\Python[version]. If selecting the location manually, a common choice is C:\Python[version] because it avoids spaces in the path, and all users can access it. Due to administrative rights, both paths may cause issues during package installation.
Other advanced options include creating shortcuts, file associations, and adding Python to PATH.

Q-2(A) Answer the following: **04**

1. **What is Break?**
   Answer: 'Break' in Python is a loop control statement. It is used to control the sequence of the loop.
2. **What is Continue?**
   Answer: The continue keyword is used to end the current iteration in a for loop (or a while loop), and continues to the next iteration.
3. **What is Pass?**
   Answer: The pass statement is used as a placeholder for future code.
4. **What is Slicing?**
   Answer: A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end.

Q-2(B) Attempt Any One (Out of Two): **02**

1. **Explain: Indexing.**
   Answer: In Python, indexing refers to the process of accessing a specific element in a sequence, such as a string or list, using its position or index number. Indexing in Python starts at 0, which means that the first element in a sequence has an index of 0, the second element has an index of 1, and so on.
2. **Explain: Array.**
   Answer: An array is a special variable, which can hold more than one value at a time. If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:
   ```
   car1 = "Ford"
   car2 = "Volvo"
   car3 = "BMW"
   ```

Q-2(C) Attempt Any One (Out of Two): **03**

1. **Explain: IF Statement.**
   Answer: If-Else statements in Python are part of conditional statements, which decide the control of code. As you can notice from the name If-Else, you can notice the code has two ways of directions.
   There are situations in real life when we need to make some decisions and based on these decisions, we decide what we should do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code.
   Conditional statements in Python languages decide the direction(Control Flow) of the flow of program execution.
   ```
   #if syntax Python
   if condition:
     # Statements to execute if
     # condition is true
   ```
2. **Explain: While loop.**
   Answer: Python While Loop is used to execute a block of statements repeatedly until a given condition is satisfied. When the condition becomes false, the line immediately after the loop in the program is executed. While loop falls under the category of indefinite iteration. Indefinite iteration means that the number of times the loop is executed isn't specified explicitly in advance.
   ```
   # Python program to illustrate
   # while loop
   count = 0
   while (count < 3):
           count = count + 1
           print("Hello Geek")
   ```

Q-2(D) Attempt Any One (Out of Two): **05**

1. **Explain: Conditional Statements with Example.**

Answer: As the name suggests, a conditional statement is used to handle conditions in your program. These statements guide the program while making decisions based on the conditions encountered by the program. In this article, we will explore three conditional statements in Python: if statement, if-else statements, if-elif-else ladder. Like every other programming language, Python also has some predefined conditional statements. A conditional statement as the name suggests itself, is used to handle conditions in your program. These statements guide the program while making decisions based on the conditions encountered by the program.

if Statement

The if statement is a conditional statement in Python used to determine whether a block of code will be executed. If the program finds the condition defined in the if statement true, it will execute the code block inside the if statement.

Syntax:

```
if condition:
    # execute code block
```

To better understand this, take a look at the below example.

```
# list of numbers
list_of_numbers = [2,4,6,9,5]
# for loop to iterate through the list and check each elements of the list
for i in list_of_numbers:
    # check if no. is odd
    if i % 2 != 0:
        # if condition is True print "odd"
        print (i, "is an odd number")
    # check if no. is even
    if i % 2 == 0:
        # if condition is false print "even"
        print (i, "is an even number")
```

if-else Statement

As discussed above, the if statement executes the code block when the condition is true. Similarly, the else statement works in conjuncture with the if statement to execute a code block when the defined if condition is false.

Syntax:

```
if condition:
    # execute code if condition is true
else:
    # execute code if condition if False
```

Now, let's expand on this concept in the previous example.

Python Example: Python program to check if a number is odd or even.

```
# list of numbers
list_of_numbers = [2,4,6,9,5]
# for loop to iterate through the list and check each element of the list
for i in list_of_numbers:
    # check if no. is odd
    if i%2 != 0:
        # if condition is True print "odd"
        print(i, "is an odd number")
    # if the no. is not odd then the no. is even therfore print "even"
    else:
        # if condition is True print "even"
        print(i, "is an even number")
```

if-elif-else ladder

The elif statement is used to check for multiple conditions and execute the code block within if any of the conditions are evaluated to be true.

The elif statement is similar to the else statement in that it is optional. Still, unlike the else statement, multiple elif statements can be in a block of code following an if statement.

```
if condition1:
   # execute this statement
elif condition2:
   # execute this statement
else:
```

```
# if non of the above conditions evaluate to True execute this statement
```

Take a look at the below example for context.

Python Example: Python program to check if a string matches the condition.

```
# take a sample string
string ="ShikshaOnline"
# check if value in the string variable
# matches "Shik"
if string == "Shik":
   print ("The first condition is true")
# check if value in the string variable
# matches "Shiksha"
elif string == "Shiksha":
   print("The second condition is true")
# check if value in the string variable
# matches "Online"
elif string == "Online":
   print("The third condition is true")
# check if value in the string variable
# matches "ShikshaOnline"
elif string=="ShikshaOnline":
   print("The fourth condition is true")
# if none of the above
# conditions evaluate to true
# execute the code inside the else block
else:
   print ("All the above conditions are false")
```

2. **Explain: Looping Structures with Example.**

Answer: A loop is a control flow statement in Python that allows you to execute a piece of code repeatedly until a specific condition is met. Loops are necessary for operations that require repetitive execution, such as iterating through a Python list of items or doing calculations many times.

Different Types of Loops in Python

To understand the Loop Structures in Python, you need to know that there are 3 Types of Loops in Python, those are:

A. Python while loop
B. Python for loop
C. Python nested loop

A. Python while loop

In Python, a while loop is a control flow statement that allows you to execute a block of code repeatedly while a given condition is true. The condition is evaluated at the start of each loop iteration, and if it is true, the loop body is executed. The loop is terminated if the condition is false.

```
count = 0
while (count < 10):
 print ('The count is:', count)
 count = count + 1
print ("Good bye!")
```

B. Python for loop

A for loop is a control flow statement in Python that allows you to iterate over a sequence of elements such as a list, tuple, or string. On each iteration, the loop variable in Python will take on the value of the next item in the sequence.

```
for letter in 'Python': # First Example
 print ('Current Letter :', letter)
fruits = ['guava', 'apple', 'mango']
for fruit in fruits: # Second Example
 print ('Current fruit :', fruit)
print ("Good bye!")
```

C. Python Nested loop

In Python, a nested loop is a loop that is contained within another loop. This indicates that the inner loop is executed within each outer loop iteration. Nested loops are handy for iterating over a two-dimensional array or processing hierarchical data.

```
i = 2
while(i < 100):
 j = 2
 while(j <= (i/j)):
   if not(i%j): break
   j = j + 1
 if (j > i/j) : print (i, " is prime")
 i = i + 1
print ("Good bye!")
```

Q-3(A) Answer the following: **04**

1. **What is List?**
   Answer: Lists are used to store multiple items in a single variable.
2. **What is Tuple?**
   Answer: Tuples are used to store multiple items in a single variable.
3. **What is Dictionary?**
   Answer: Dictionaries are used to store data values in key:value pairs
4. **What is Sorting?**
   Answer: It sorts the list in ascending order.

Q-3(B) Attempt Any One (Out of Two): **02**

1. **How to Define List?**
   Answer: To create a list in Python, write a set of items within square brackets ([]) and separate each item with a comma. Items in a list can be any basic object type found in Python, including integers, strings, floating point values or boolean values.
   ```
   z = [3, 7, 4, 2]
   ```
2. **How to Define Tuple?**
   Answer: Tuples are used to store multiple items in a single variable. Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage. A tuple is a collection which is ordered and unchangeable. Tuples are written with round brackets.
   ```
   thistuple = ("apple", "banana", "cherry")
   print(thistuple)
   ```

Q-3(C) Attempt Any One (Out of Two): **03**

1. **What is Dictionary? Explain with Example.**
   Answer: Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is ordered*, changeable and do not allow duplicates. Dictionary items are ordered, changeable, and do not allow duplicates. Dictionary items are presented in key:value pairs, and can be referred to by using the key name. When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change. Unordered means that the items do not have a defined order, you cannot refer to an item by using an index.
   ```
   thisdict = {
     "brand": "Ford",
     "model": "Mustang",
     "year": 1964
   }
   print(thisdict)
   ```
2. **What is List? Explain with Example.**
   Answer: Lists are used to store multiple items in a single variable. Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage. Lists are created using square brackets. List items are ordered, changeable, and allow duplicate values. List items are indexed, the first item has index [0], the second item has index [1] etc. When we say that lists are ordered, it means that the items have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list.
   ```
   thislist = ["apple", "banana", "cherry", "apple", "cherry"]
   print(thislist)
   ```

Q-3(D) Attempt Any One (Out of Two): 05

1. **Write a Program for Converting List to Dictionary.**

   Answer:

```
import itertools
def convert(lst):
    pairs = itertools.zip_longest(*[iter(lst)] * 2, fillvalue=None)
    dct = {key: value for key, value in pairs}
    return dct
lst = ['a', 1, 'b', 2, 'c', 3]
print(convert(lst))
```

2. **Write a Program for Converting Strings to Dictionary.**

   Answer:

```
# Python3 code to demonstrate
# convert dictionary string to dictionary
# using json.loads()
import json
# initializing string
test_string = '{"Nikhil" : 1, "Akshat" : 2, "Akash" : 3}'
# printing original string
print("The original string : " + str(test_string))
# using json.loads()
# convert dictionary string to dictionary
res = json.loads(test_string)
# print result
print("The converted dictionary : " + str(res))
```

Q-4(A) Answer the following: 04

1. **What is Error?**
   Answer: An error is an action which is inaccurate or incorrect. In some usages, an error is synonymous with a mistake.
2. **What is Exception?**
   Answer: An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
3. **What is Module?**
   Answer: A module is a Python object with arbitrarily named attributes that you can bind and reference.
4. **What are Generators?**
   Answer: In Python, a generator is a function that returns an iterator that produces a sequence of values when iterated over.

Q-4(B) Attempt Any One (Out of Two): 02

1. **Explain: Anonymous Functions.**
   Answer: Python Lambda Functions are anonymous functions means that the function is without a name. As we already know the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.
2. **What is Variable Length Argument in Functions?**
   Answer: Variable-length arguments refer to a feature that allows a function to accept a variable number of arguments in Python. It is also known as the argument that can also accept an unlimited amount of data as input inside the function. There are two types in Python: Non – Keyworded Arguments (*args) , Keyworded Arguments (**kwargs)

Q-4(C) Attempt Any One (Out of Two): 03

1. **Explain: Functional Decorators with Example.**
   Answer: Decorators are a very powerful and useful tool in Python since it allows programmers to modify the behaviour of a function or class. Decorators allow us to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it. But before diving deep into decorators let us understand some concepts that will come in handy in learning the decorators.

```
# Python program to illustrate functions
# can be treated as objects
def shout(text):
    return text.upper()
```

```
print(shout('Hello'))
yell = shout
print(yell('Hello'))
```

2. **Explain: Math Module with Example.**

   Answer: Math Module consists of mathematical functions and constants. It is a built-in module made for mathematical tasks. The math module provides the math functions to deal with basic operations such as addition(+), subtraction(-), multiplication(*), division(/), and advanced operations like trigonometric, logarithmic, and exponential functions. In this article, we learn about the math module from basics to advanced, we will study the functions with the help of good examples.

```
import math
r = 4
pie = math.pi
print(pie * r * r)
```

Q-4(D) Attempt Any One (Out of Two): 05

1. **Write a Program to Open and Close text file.**

   Answer:

```
# open the file using open() function
file = open("sample.txt")
# Reading from file
print(file.read())
# Reading from file
print(file.read())
# closing the file
file.close()
```

2. **Write a Program for Zipping and Unzipping files.**

   Answer:

```
# importing required modules
from zipfile import ZipFile
import os
def get_all_file_paths(directory):
    # initializing empty file paths list
    file_paths = []
    # crawling through directory and subdirectories
    for root, directories, files in os.walk(directory):
        for filename in files:
            # join the two strings in order to form the full filepath.
            filepath = os.path.join(root, filename)
            file_paths.append(filepath)
    # returning all file paths
    return file_paths
def main():
    # path to folder which needs to be zipped
    directory = './python_files'
    # calling function to get all file paths in the directory
    file_paths = get_all_file_paths(directory)
    # printing the list of all files to be zipped
    print('Following files will be zipped:')
    for file_name in file_paths:
        print(file_name)
    # writing files to a zipfile
with ZipFile('my_python_files.zip','w') as zip:
    # writing each file one by one
    for file in file_paths:
        zip.write(file)
    print('All files zipped successfully!')
    if __name__ == "__main__":
        main()
```

Q-5(A) Answer the following: **04**

1. **Full Form: OOP.**
   Answer: Object-oriented programming.
2. **Full Form: MRO.**
   Answer: Method Resolution Order.
3. **What is Class?**
   Answer: In object-oriented programming, a class is an extensible program-code-template for creating objects, providing initial values for state and implementations of behavior.
4. **What is Object?**
   Answer: In computer science, an object can be a variable, a data structure, a function, or a method. As regions of memory, they contain value and are referenced by identifiers.

Q-5(B) Attempt Any One (Out of Two): **02**

1. **What is Polymorphism?**
   Answer: The word polymorphism means having many forms. In programming, polymorphism means the same function name (but different signatures) being used for different types. The key difference is the data types and number of arguments used in function.
2. **What is Inheritance?**
   Answer: Inheritance allows us to define a class that inherits all the methods and properties from another class.
   Parent class is the class being inherited from, also called base class. Child class is the class that inherits from another class, also called derived class.

Q-5(C) Attempt Any One (Out of Two): **03**

1. **Explain: Operator Overloading.**
   Answer: Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example, operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because '+' operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading.

```
# Python program to show use of
# + operator for different purposes.
print(1 + 2)
# concatenate two strings
print("Geeks"+"For")
# Product two numbers
print(3 * 4)
# Repeat the String
print("Geeks"*4)
```

2. **Explain: Interfaces in Python.**
   Answer: In object-oriented languages like Python, the interface is a collection of method signatures that should be provided by the implementing class. Implementing an interface is a way of writing an organized code and achieve abstraction. The package zope.interface provides an implementation of "object interfaces" for Python. It is maintained by the Zope Toolkit project. The package exports two objects, 'Interface' and 'Attribute' directly. It also exports several helper methods. It aims to provide stricter semantics and better error messages than Python's built-in abc module.

```
import zope.interface
class MyInterface(zope.interface.Interface):
        x = zope.interface.Attribute("foo")
        def method1(self, x):
                pass
        def method2(self):
                pass
print(type(MyInterface))
print(MyInterface.__module__)
print(MyInterface.__name__)
# get attribute
x = MyInterface['x']
print(x)
print(type(x))
```

Q-5(D) Attempt Any One (Out of Two): **05**

1. **Differentiate: Abstract Classes V/S Interfaces with Example.**

Answer: **What is an Abstract class in Python?**

A blueprint for other classes might be thought of as an abstract class. You may use it to define a collection of methods that are required for all subclasses derived from the abstract class. An abstract class is one that includes one or more abstract methods. A method that has a declaration but no implementation is said to be abstract. We use an abstract class for creating huge functional units. An abstract class is used to offer a standard interface for various implementations of a component.

```
from abc import ABC, abstractmethod
class Animal(ABC):
    def move(self):
        pass
class Human(Animal):
    def move(self):
        print("I can walk and run")
class Snake(Animal):
    def move(self):
        print("I can crawl")
class Dog(Animal):
    def move(self):
        print("I can bark")
class Lion(Animal):
    def move(self):
        print("I can roar")
# Driver code
R = Human()
R.move()
K = Snake()
K.move()
R = Dog()
R.move()
K = Lion()
K.move()
```

What is an Interface in Python?

The interface in object-oriented languages like Python is a set of method signatures that the implementing class is expected to provide. Writing ordered code and achieving abstraction are both possible through interface implementation.

```
import zope.interface
class MyInterface(zope.interface.Interface):
    x = zope.interface.Attribute("foo")
    def method1(self, x):
        pass
    def method2(self):
        pass
print(type(MyInterface))
print(MyInterface.__module__)
print(MyInterface.__name__)
# get attribute
x = MyInterface['x']
print(x)
print(type(x))
```

2. **Differentiate: Method Overriding V/S Method Overloading with Example.**

Answer: **Method Overriding:**

Method overriding is an ability of any object-oriented programming language that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class. The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

```
# Python program to demonstrate
# method overriding
# Defining parent class
class Parent():
    # Constructor
    def __init__(self):
        self.value = "Inside Parent"
    # Parent's show method
    def show(self):
        print(self.value)
# Defining child class
class Child(Parent):
    # Constructor
    def __init__(self):
        self.value = "Inside Child"
    # Child's show method
    def show(self):
        print(self.value)
# Driver's code
obj1 = Parent()
obj2 = Child()
obj1.show()
obj2.show()
```

Method Overloading:

Two or more methods have the same name but different numbers of parameters or different types of parameters, or both. These methods are called overloaded methods and this is called method overloading. Like other languages (for example, method overloading in C++) do, python does not support method overloading by default. But there are different ways to achieve method overloading in Python. The problem with method overloading in Python is that we may overload the methods but can only use the latest defined method.

```
# First product method.
# Takes two argument and print their
# product
def product(a, b):
    p = a * b
    print(p)
# Second product method
# Takes three argument and print their
# product
def product(a, b, c):
    p = a * b*c
    print(p)
# Uncommenting the below line shows an error
# product(4, 5)
# This line will call the second product method
product(4, 5, 5)
```