PADHLE BHAI.IN

(MTE-2) Question Bank

Wtih Answers

Q) What is String Slicing?

String slicing is a way to grab a part of a word or sentence in Python. It's like cutting a piece of cake.

Q) How to Do String Slicing?

Use square brackets [] with a colon : inside them.

string[start:stop] gets characters from start to stop-1.

a) Encapsulation:

Encapsulation is like putting things in a box. In programming, it means putting the data (variables) and the methods (functions) that work on the data inside a single unit, often called a class. It helps organise and protect the data by controlling access to it.

Sample Example:

python Copy code

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

    def bark(self):
        print("Woof!")

# Creating an instance (object) of the Dog class
my_dog = Dog("Buddy", 3)

# Accessing data (encapsulated variables)
print(my_dog.name)  # Output: Buddy

# Calling a method (encapsulated function)
my_dog.bark()  # Output: Woof!
```

b) Iteration:

Iteration is like doing something repeatedly. In programming, it means repeatedly executing a set of statements. It's often used for going through each item in a collection (like a list) or for repeating a block of code a certain number of times.

Sample Example:

```python
# Iterating over a list
fruits = ["apple", "banana", "orange"]
for fruit in fruits:
    print(fruit)

# Output:
# apple
# banana
# orange
```

c) Data Abstraction:

Data abstraction is like using something without knowing how it works inside. In programming, it means hiding the complex details and showing only the essential parts. It allows you to use a tool (like a function or class) without needing to understand its internal workings.

```python
# Using a function without knowing its implementation details
def add_numbers(a, b):
    return a + b

# Using the function
result = add_numbers(5, 3)
print(result)  # Output: 8
```

In this example, you don't need to know how add_numbers adds the numbers; you just use it for its intended purpose. This is an example of data abstraction.

Q) What is meant by key-value pairs in the Dictionary?

Key-value pairs in a dictionary are like words and their meanings in a real dictionary. In Python, a dictionary is a collection of data where each piece of data has a label, or "key," and a

corresponding "value." You use the key to look up or access its associated value.

```python
# Creating a dictionary with key-value pairs
person = {
    "name": "John",
    "age": 25,
    "city": "New York"
}

# Accessing values using keys
print(person["name"])  # Output: John
print(person["age"])   # Output: 25
print(person["city"])  # Output: New York
```

In this example:

"name", "age", and "city" are keys.
"John", 25, and "New York" are their corresponding values.

Just like you'd look up a word in a dictionary to find its meaning, you use the key to find the associated value in a Python dictionary. Each key in a dictionary must be unique, but values can be duplicated.

Q) Differentiate between a) break and continue statements b) local & global variables?

a) Break and Continue Statements:

Break statement:

Purpose: It is used to exit (break out of) a loop prematurely.
Effect: When encountered, the loop immediately stops, and the program continues with the next statement after the loop.

Example:

```
for i in range(5):
    if i == 3:
        break
    print(i)
# Output: 0, 1, 2
```

Continue statement:

Purpose: It is used to skip the rest of the code inside a loop for the current iteration and move to the next iteration.
Effect: When encountered, the loop jumps to the next iteration, skipping the remaining code in the loop block.

Example:

```
for i in range(5):
    if i == 2:
        continue
    print(i)
# Output: 0, 1, 3, 4
```

b) Local and Global Variables:

Local Variable:

Scope: It is declared inside a function and is only accessible within that function.
Lifetime: It exists as long as the function is executing, and its memory is freed when the function exits.

Example:

```
def my_function():
    x = 10  # Local variable
    print(x)

my_function()
# Output: 10
# print(x)  # This would result in an error because x is local to the function.
```

Global Variable:

Scope: It is declared outside any function and can be accessed throughout the entire program.
Lifetime: It exists until the program terminates or until it is explicitly deleted.

Example:

```
y = 5  # Global variable

def another_function():
    print(y)

another_function()
# Output: 5
print(y)  # Accessing the global variable outside the function.
```

In summary, **break** is used to exit a loop, and **continue** is used to skip the rest of the loop code for the current iteration. Local variables are confined to the scope of the function where they are declared, while global variables can be accessed throughout the entire program.

Q) What are the various parameters in functions?

In Python functions, parameters are the inputs that a function can accept. There are different types of parameters:

1) Positional Parameters:

Definition: These are the most common type of parameters, and the values are passed based on their position or order.

Example:

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

result = add_numbers(3, 5)
# Here, 3 is assigned to x, and 5 is assigned to y.
```

2)Keyword Parameters:

Definition: Values are passed using the parameter names, which allows you to specify which argument corresponds to which parameter.

Example:

```
def greet(name, age):
    print("Hello, {}! You are {} years old.".format(name, age))

greet(age=25, name="Alice")
# Here, the values are assigned based on the parameter names.
```

3)Default Parameters:

Definition: Parameters can have default values, and if a value is not provided when the function is called, the default value is used.

Example:

```
def power(base, exponent=2):
    return base ** exponent

result1 = power(3)    # Uses the default exponent value (2)
result2 = power(3, 4)  # Uses the provided exponent value (4)
```

4)Variable-Length Parameters:

Definition: Functions can accept a variable number of arguments.

Two types:
*__args__ for variable positional parameters.
**__kwargs__ for variable keyword parameters.

Example:

```
def print_arguments(*args, **kwargs):
    print("Positional arguments:", args)
    print("Keyword arguments:", kwargs)

print_arguments(1, "two", 3.0, name="Alice", age=25)
# Accepts any number of positional and keyword arguments.
```

These are the primary types of parameters in Python functions. Understanding how to use them gives you flexibility and control when designing and calling functions.

Q) Discuss the pass statement.

The **pass statement** in Python is a null operation or a no-op. It serves as a placeholder where syntactically some code is required, but you don't want to execute any statements. It is often used when a statement is required by Python syntax (like in a loop or an if statement), but you don't want to take any action.

```
def my_function():
    pass  # This function doesn't do anything yet

for i in range(5):
    if i == 3:
        pass  # This block doesn't do anything for i=3
    else:
        print(i)
```

In this example:

my_function is defined but does nothing (it contains only the **pass** statement).

In the for loop, when **i** is equal to **3**, the **pass** statement is used in the **if** block to indicate that no action should be taken for this particular case.

The **pass** statement is helpful in situations where you need to include a statement for syntactic reasons, but the actual implementation or code is not yet available or necessary. It allows you to create a minimal valid structure without executing any specific instructions.

Q) Explain the while loop construct with syntax and flowchart.Write a program to calculate the reverse of a number using a while loop.

While Loop Explanation:

Syntax:

```
while condition:
    # Code to be repeated as long as the condition is True
    # ...
    # (Don't forget to update the condition to avoid an infinite loop)
```

Flowchart:

- The loop starts by checking the condition.
- If the condition is True, the code block inside the loop is executed.
- After executing the code block, the condition is checked again.
- This process continues until the condition becomes False, at which point the loop ends.

Program to Calculate the Reverse of a Number using While Loop:

```python
# Function to calculate the reverse of a number
def reverse_number(number):
    reversed_number = 0
    while number > 0:
        digit = number % 10
        reversed_number = reversed_number * 10 + digit
        number = number // 10
    return reversed_number

# Input
original_number = int(input("Enter a number: "))

# Calculate and display the reverse
result = reverse_number(original_number)
print(f"The reverse of {original_number} is: {result}")
```

```
Enter a number: 12345
The reverse of 12345 is: 54321
```

How to create a dictionary in python. Explain adding & modifying dictionary values with suitable examples.

Creating a Dictionary:
To create a dictionary in Python, use curly braces {} and separate key-value pairs with colons :

For example:

```python
my_dict = {"name": "John", "age": 25, "city": "New York"}
```

Adding Values:

To add a new key-value pair, use square brackets [] with the new key and assign the corresponding value.

```
my_dict["gender"] = "Male"
```

Modifying Values:
To modify the value of an existing key, use square brackets [] with the key and assign the new value.

```
my_dict["age"] = 26
```

Example:

```
person = {"name": "Alice", "age": 30, "city": "London"}
person["occupation"] = "Engineer"
person["age"] = 31
```

Q) Explain conditional statements in detail with examples (if, if..else, if..elif..else).

1. if Statement:

Definition: The if statement is used to execute a block of code if a certain condition is true.

```
x = 10
if x > 5:
    print("x is greater than 5")
```

2. if...else Statement:

Definition: The if...else statement allows you to execute one block of code if the condition is true and another block if it's false.

Example:

```
y = 3
if y % 2 == 0:
    print("y is even")
else:
    print("y is odd")
```

In this example, the print statement in the else block will be executed because the condition y % 2 == 0 is false.

3. if...elif...else Statement:

Definition: The if...elif...else statement is used when you have multiple conditions to check. It allows you to check each condition one by one and execute the corresponding block of code for the first true condition.

Example:

```
grade = 75
if grade >= 90:
    print("A")
elif grade >= 80:
    print("B")
elif grade >= 70:
    print("C")
else:
    print("Fail")
```

In this example, based on the value of grade, one of the conditions will be true, and the corresponding print statement will be executed.

Q) How do we create a Class and Object of a Class in Python? Also differentiate between Object Oriented Programming and Procedure Oriented Programming.

Creating a Class and Object in Python:

1. Creating a Class:

In Python, you create a class using the class keyword, followed by the class name and a colon. Inside the class, you define attributes (characteristics) and methods (functions).

Example:

```
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"{self.make} {self.model}")
```

In this example, the Car class has attributes make and model and a method display_info to print information about the car.

2. Creating an Object:

To create an object (instance) of a class, you call the class as if it were a function, passing any required arguments to its **__init__** method.

Example:

```
my_car = Car(make="Toyota", model="Camry")
```

Object-Oriented Programming (OOP) Procedure-Oriented Programming (POP):

Object-Oriented Programming (OOP):

Definition: Object-Oriented Programming is a programming paradigm that organises code into reusable and self-contained objects. Objects represent instances of classes, which encapsulate data and behaviour.

Key Concepts:

- **Classes and Objects:** Code is organised into classes, and objects are instances of these classes.
- **Encapsulation:** Bundling data and methods that operate on the data into a single unit (class).
- **Inheritance:** A mechanism where a new class can inherit properties and behaviours from an existing class.
- **Polymorphism:** Objects can take on multiple forms; the same method can behave differently in different classes.

```
class Animal:
    def sound(self):
        pass  # Abstract method

class Dog(Animal):
    def sound(self):
        return "Woof!"

class Cat(Animal):
    def sound(self):
        return "Meow!"

my_dog = Dog()
my_cat = Cat()

print(my_dog.sound())  # Output: Woof!
print(my_cat.sound())  # Output: Meow!
```

Key Difference:

OOP focuses on objects: OOP organises code around objects that encapsulate data and behaviour, promoting reusability and modularity.

POP focuses on procedures/functions: POP organises code around procedures or functions that operate on shared data, leading to sequential execution.

Q) Discuss the features of OOPs in Python.

1. Classes and Objects:

Definition: Classes are user-defined blueprints that encapsulate data (attributes) and methods (functions) that operate on that data.

Example:

```
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"{self.make} {self.model}")

my_car = Car(make="Toyota", model="Camry")
```

2. Encapsulation:

Definition: Encapsulation is the bundling of data and methods that operate on the data into a single unit (class). It hides the internal details of the object.

Example:

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

    def display_info(self):
        print(f"Name: {self.name}, Age: {self.age}")

student = Student(name="Alice", age=20)
student.display_info()
```

3. Inheritance:

Definition: Inheritance allows a new class (subclass/derived class) to inherit attributes and methods from an existing class (superclass/base class).

Example:

```
class Animal:
    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Woof!"

class Cat(Animal):
    def sound(self):
        return "Meow!"
```

4. Polymorphism:

Definition: Polymorphism allows objects of different classes to be treated as objects of a common base class. It enables a single interface to represent different types.

Example:

```python
class Animal:
    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Woof!"

class Cat(Animal):
    def sound(self):
        return "Meow!"

def make_sound(animal):
    return animal.sound()

my_dog = Dog()
my_cat = Cat()

print(make_sound(my_dog))  # Output: Woof!
print(make_sound(my_cat))  # Output: Meow!
```

5. Abstraction:

Definition: Abstraction is the process of simplifying complex systems by modelling classes based on their essential properties and behaviours.

Example:

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2
```

6. Modularity:

Definition: Modularity allows code to be divided into small, independent units (classes) that can be developed, tested, and maintained separately.

Example:

```
class Calculator:
    def add(self, x, y):
        return x + y

    def subtract(self, x, y):
        return x - y
```

Q) Explain various types of function parameters in Python with suitable examples.

1. Positional Parameters:

Definition: These parameters are passed to a function based on their position in the function call.
Example:

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

result = add(3, 5)  # Here, 3 is assigned to x, and 5 is assigned to y
```

2. Keyword Parameters:

Definition: Values are passed using the parameter names, allowing you to specify which argument corresponds to which parameter.
Example:

```
def greet(name, age):
    print(f"Hello, {name}! You are {age} years old.")

greet(age=25, name="Alice")  # Here, the values are assigned based on the parameter
```

Q) Differentiate between OOP & POP

Abstraction Level:

OOP: Emphasises a higher level of abstraction by modelling entities using objects.
POP: Works at a lower level of abstraction, emphasising procedures and functions.

Data and Code Relationship:

OOP: Encourages bundling data and methods together in a class, emphasising the relationship between data and code.
POP: Typically separates data and code, with functions operating on shared data.

Code Reusability:

OOP: Promotes code reusability through inheritance and polymorphism.
POP: Code reusability is achieved through functions, but it may be more limited.

Flexibility and Extensibility:
OOP: Offers greater flexibility and extensibility through features like inheritance.
POP: May require more effort for extending or modifying code.

Q) Explain range function in Python. Write a program to print the following pattern:

```
*
**
***
****
*****
```

range Function in Python:
The range function in Python is used to generate a sequence of numbers. It can be used in different ways:

range(stop): Generates numbers from 0 to stop-1.
range(start, stop): Generates numbers from start to stop-1.
range(start, stop, step): Generates numbers from start to stop-1 with a step size of step.

Example:

```python
# Using range to generate a sequence
for i in range(5):
    print(i)
# Output: 0, 1, 2, 3, 4
```

Program to Print the Pattern:
Now, let's use the range function to print the pattern you mentioned:

```python
def print_pattern(rows):
    for i in range(1, rows + 1):
        for j in range(i):
            print("*", end=" ")
        print()

# Call the function with the desired number of rows
print_pattern(4)
```

Output:

```
*
* *
* * *
* * * *
```

Q) Explain the for loop construct with syntax and flowchart.Write a program to print the following pattern:

1
12
1 2 3
1 2 3 4
1 2 3 4 5

for Loop in Python:

Syntax:

```
for variable in sequence:
    # Code to be executed in each iteration
    # ...
```

variable: Variable that takes the value of the next item in the sequence in each iteration.

sequence: Iterable object (list, tuple, string, etc.) that the loop iterates over.

- The loop starts by initialising the variable with the first value in the sequence.
- The condition is checked in each iteration. If it's true, the code block is executed.
- After the code block, the loop moves to the next iteration, updating the variable.
- This process continues until the condition becomes false.

Program to Print the Pattern:

```python
def print_pattern(rows):
    for i in range(1, rows + 1):
        for j in range(1, i + 1):
            print(j, end=" ")
        print()

# Call the function with the desired number of rows
print_pattern(5)
```

Q) What is Inheritance? State all its types with suitable diagrams. Explain any one with the help of a program in python.

Inheritance in Python:

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a new class (subclass/derived class) to inherit attributes and methods from an existing class (superclass/base class). This promotes code reusability and establishes a relationship between the classes.

Types of Inheritance:

1. **Single Inheritance**:A subclass inherits from only one superclass.
2. **Multiple Inheritance:**A subclass inherits from more than one superclass.
3. **Multilevel Inheritance**:A subclass serves as a superclass for another class.
4. **Hierarchical Inheritance**:Multiple classes inherit from a single superclass.

Example: Single Inheritance in Python:

```python
# Superclass
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} makes a sound")

# Subclass inheriting from Animal
class Dog(Animal):
    def bark(self):
        print(f"{self.name} barks")

# Creating an object of the subclass
my_dog = Dog(name="Buddy")

# Using methods from both classes
my_dog.speak()  # Inherited from Animal
my_dog.bark()   # Defined in Dog
```

PADHLE BHAI.IN

(MTE-2) Question Bank

Wtih Answers