CHAPTER -5

Getting Started with Python

2MARK QUESTIONS

Q1: What is Python, and why is it popular for beginners in programming?
Answer:

Python is a high-level, interpreted programming language known for its readability and simplicity. It is popular among beginners due to its easy syntax, extensive libraries, and versatility, making it suitable for various applications.

Q2: Explain the difference between Python 2 and Python 3. (2 marks)

Answer:

Python 2 and Python 3 are two major versions of the Python programming language. Python 3 is the latest version, designed to address certain shortcomings of Python 2. The key differences include print syntax, Unicode support, and integer division.

Q3: What is the purpose of indentation in Python?

Answer:

In Python, indentation is used to define the structure of code blocks. It replaces traditional curly braces or keywords, making the code more readable. Proper indentation is crucial for indicating the beginning and end of loops, conditionals, and functions.

Q4: Describe the role of a variable in Python. Provide an example.

Answer:

A variable in Python is a named storage location used to store data. It allows the programmer to manipulate and reference values by a specific name. For example:

age = 25

Question 5: What is a Python function, and how is it defined?

Answer:

A Python function is a reusable block of code that performs a specific task. It is defined using the def keyword, followed by the function name, parameters, and a colon. For example:

```
def greet(name):
    print ("Hello, " + name + "!")
```

Q6: Explain the purpose of the 'if' statement in Python. Provide an example. Answer:

The 'if' statement in Python is used for conditional execution. It allows the program to make decisions based on whether a certain condition is true or false. For example:

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

Q 7: What is a Python list, and how is it different from a tuple?

Answer:

A list and a tuple are both data structures in Python used to store ordered collections of items. The key difference is that lists are mutable (can be modified), while tuples are immutable (cannot be modified after creation).

Question 8: Explain the purpose of a Python module. Provide an example.
Answer:

A Python module is a file containing Python code that can be executed or reused. It allows for better organization and reusability of code. For example, if the file math_operations.py contains mathematical functions:

```
# math_operations.py
def add(x, y):
    return x + y
def subtract(x, y):
    return x - y
```

It can be used in another script:

```
# main.py

import math_operations

result = math_operations.add(5, 3)
```

Q 9: How are comments added to Python code?

Answer:

Comments in Python are added using the # symbol. Anything after the # on a line is treated as a comment and is not executed. For example:

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

Q10: What is the purpose of the 'for' loop in Python? Provide an example.
Answer:

The 'for' loop in Python is used to iterate over a sequence (such as a list, tuple, or string). It simplifies the process of repeatedly executing a block of code. For example:

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

4MARK QUESTIONS

Q1: Explain the concept of object-oriented programming (OOP) and provide an example in Python.

Answer:

Object-oriented programming is a programming paradigm that uses objects, which encapsulate data and behavior, to model real-world entities. In Python, classes and objects facilitate OOP. Example:

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
 def bark(self):
        print(f"{self.name} says Woof!")
my_dog = Dog("Buddy", 3)
my_dog.bark()
```

Q2: Describe the purpose of the 'try-except' block in Python and provide an example of its usage.

Answer:

The 'try-except' block is used for exception handling in Python. It allows the programmer to handle potential errors gracefully. Example:

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

Q3: Discuss the differences between a function and a method in Python. Provide examples.

Answer:

In Python, a function is a standalone block of code, while a method is a function associated with an object. Example:

```
# Function

def greet(name):

    print(f"Hello, {name}!")

greet("Alice")

# Method

class Person:

    def greet(self, name):

        print(f"Hello, {name}!")

person = Person()

person.greet("Bob")
```

Q 4: Explain the role of the 'else' clause in a Python 'try-except' block. Provide an example.

Answer:

The 'else' clause in a 'try-except' block is executed if no exceptions occur. Example:

```
try:

    result = 10 / 2

except ZeroDivisionError:

    print("Cannot divide by zero!")

else:

    print(f"Result: {result}")
```

Q5: Discuss the purpose of the 'with' statement in Python, especially in the context of file handling. Provide an example.

Answer:

The 'with' statement simplifies resource management, ensuring that resources are properly handled. In file handling, it is often used to automatically close files. Example:

```
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

Q6. Compare and contrast the usage of 'append()' and 'extend()' methods in Python lists. Provide examples.

Answer:

Both 'append()' and 'extend()' methods are used with lists. 'append()' adds an element to the end of the list, while 'extend()' adds elements from an iterable. Example:

```
# append()
numbers = [1, 2, 3]
numbers.append(4)  # Result: [1, 2, 3, 4]
# extend()
numbers = [1, 2, 3]
numbers.extend([4, 5])  # Result: [1, 2, 3, 4, 5]
```

Q7: Elaborate on the concept of a Python decorator. Provide an example of creating and using a decorator.

Answer:

A decorator is a function that modifies the behavior of another function. Example:

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper
@my_decorator
def say_hello():
    print("Hello!")
say_hello()
```

Q8: Discuss the purpose of the 'map()' function in Python. Provide an example of its usage.

Answer:

The 'map()' function applies a given function to all items in an input iterable and returns an iterator. Example:

```
numbers = [1, 2, 3, 4]
squared = map(lambda x: x**2, numbers)
print(list(squared))  # Result: [1, 4, 9, 16]
```

Q9: Explain the concept of a Python generator and provide an example of creating and using a generator.

Answer:

A generator is a special type of iterable, created using a function with 'yield' statements. Example:

```
def square_numbers(n):
    for i in range(n):
        yield i**2
squares = square_numbers(5)
print(list(squares))  # Result: [0, 1, 4, 9, 16]
```

Q10: Describe the purpose of the 'init' method in Python classes. Provide an example of a class with an 'init' method.

Answer:

The 'init' method is a special method in Python classes used for initializing object attributes. Example:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
person = Person("Alice", 25)
print(person.name)  # Result: Alice
```

Feel free to adapt these questions based on the specific content covered in your class.

10MARK QUESTIONS

Q1: Discuss the principles of DRY (Don't Repeat Yourself) and code reusability in Python. Provide examples demonstrating these principles.

Answer:

The DRY principle emphasizes avoiding code duplication for maintainability. Code reusability involves creating modular and reusable code. For example, consider a utility function to calculate the square of a number, which can be reused in multiple parts of the code:

```
def square(number):
    return number ** 2
result1 = square(5)
result2 = square(8)
```

Q2: Explain the concept of Python generators and their advantages over traditional functions. Provide an example illustrating the use of generators.

Answer:

Python generators are special functions that allow the creation of iterators using 'yield' statements. They are memory-efficient for handling large datasets. Example:

```
def square_numbers(n):
    for i in range(n):
        yield i**2
squares = square_numbers(5)
result = list(squares)  # Result: [0, 1, 4, 9, 16]
```

Q3: Discuss the differences between 'deep copy' and 'shallow copy' in Python, and provide examples demonstrating each.

Answer:

In Python, a 'deep copy' creates a new object and recursively copies the objects found in the original, while a 'shallow copy' creates a new object and inserts references to the objects found in the original. Example:

```
import copy
original_list = [[1, 2, 3], [4, 5, 6]]
# Shallow copy
shallow_copy = copy.copy(original_list)
# Deep copy
deep_copy = copy.deepcopy(original_list)
```

Q4: Elaborate on the purpose and usage of Python decorators. Provide an example demonstrating the creation and application of decorators.

Answer:

Python decorators are functions that modify the behavior of other functions. They are often used for code reusability and enhancing functionality. Example:

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper
@my_decorator
def say_hello():
    print("Hello!")
say_hello()
```

Q5: Discuss the concept of exception handling in Python. Provide examples demonstrating the use of 'try', 'except', 'else', and 'finally' blocks.

Answer:

Exception handling allows graceful handling of errors in Python. Example:

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
else:
    print(f"Result: {result}")
finally:
    print("This block always executes.")
```

Q6: Explore the application of Python modules and packages for code organization. Provide examples illustrating the creation and usage of modules and packages.

Answer:

Modules and packages in Python aid in organizing and structuring code. Example:

```
# Module: math_operations.py
def add(x, y):
    return x + y
def subtract(x, y):
    return x - y
# Package: my_package (with __init__.py file)
# Usage: from my_package import math_operations
```

Q7: Explain the concepts of 'map()', 'filter()', and 'reduce()' functions in Python. Provide examples demonstrating the use of each. Answer: These functions are used for processing iterables in a functional programming paradigm.

```
# map()
numbers = [1, 2, 3, 4]
squared = map(lambda x: x**2, numbers)  # Result: [1, 4, 9, 16]
# filter()
even_numbers = filter(lambda x: x % 2 == 0, numbers)  # Result: [2, 4]
# reduce()
from functools import reduce
product = reduce(lambda x, y: x * y, numbers)  # Result: 24
```

Q8: Discuss the principles of code testing and the use of the 'unittest' module in Python. Provide examples illustrating the creation and execution of unit tests.

Answer:

Code testing ensures the reliability of code. Example:

```
import unittest

def add(a, b):

    return a + b

class TestAddFunction(unittest.TestCase):

    def test_add_positive_numbers(self):

        result = add(2, 3)

        self.assertEqual(result, 5)

if __name__ == '__main__':

    unittest.main()
```

Q9: Describe the purpose of the 'contextlib' module in Python. Provide examples demonstrating the use of the 'contextmanager' decorator.

Answer:

The 'contextlib' module facilitates context management using the 'contextmanager' decorator. Example:

```
from contextlib import contextmanager

@contextmanager

def my_context():

    print("Entering the context.")

    yield

    print("Exiting the context.")

with my_context():

    print("Inside the context.")
```

Q10: Discuss the importance of documentation in Python programming. Provide examples of docstrings and explain how they contribute to code readability.

Answer:

Documentation enhances code understanding. Example:

```
def add(a, b):
    """
 This function adds two numbers.
 Parameters:
    a (int): The first number.
    b (int): The second number.
 Returns:
    int: The sum of the two numbers.
    """
    return a + b
```

Multiple-Choice Questions (MCQs):

Q 1: What does the acronym 'DRY' stand for in programming principles?

a) Don't Repeat Yourself

b) Do Run Yourself

c) Duplicate, Repeat, Yield

d) Develop Reusable Yieldings

Answer: a) Don't Repeat Yourself

Q 2: What is the purpose of the 'yield' keyword in Python?

a) To terminate a loop

b) To generate a random number

c) To create a generator function

d) To define a class method

Answer: c) To create a generator function

Q 3: Which method creates a 'deep copy' of a list in Python?

a) copy()

b) deepcopy()

c) clone()

d) replicate()

Answer: b) deepcopy()

Q 4: What does the 'with' statement in Python primarily assist with?

a) Loop control

b) File handling and resource management

c) Mathematical calculations

d) String manipulation

Answer: b) File handling and resource management

Q 5: In Python, what does the 'filter()' function do?

a) Filters out unwanted emails

b) Filters elements from a list based on a function's result

c) Applies a filter to an image

d) Creates a filter for network traffic

Answer: b) Filters elements from a list based on a function's result

Q 6: Which statement is true about Python decorators?

a) They can only be applied to classes

b) They modify the behavior of other functions

c) They are used for creating GUI elements

d) They are exclusively used in web development

Answer: b) They modify the behavior of other functions

Q 7: What is the purpose of the 'map()' function in Python?

a) Navigating through a dictionary

b) Mapping keys to values

c) Applying a function to each item in an iterable

d) Drawing a map with geographical data

Answer: c) Applying a function to each item in an iterable

Q 8: Which module is commonly used for writing unit tests in Python?

a) unittest

b) testingmodule

c) pytest

d) testutils

Answer: a) unittest

Q 9: What does the 'contextmanager' decorator facilitate in Python?

a) Context switching in threads

b) Context management for exceptions

c) Managing memory context

d) Handling context menus in GUIs

Answer: b) Context management for exceptions

Q 10: What is the purpose of a docstring in Python?

a) Displaying documentation in the console

b) Providing a way to document a class

c) Enforcing code style rules

d) Writing inline comments

Answer: b) Providing a way to document a class

Fill in the Blanks:

Q1: The 'init' method is called when a new __________ is created.

Answer: object or instance

Q2: The 'else' clause in a 'try-except' block is executed when __________ exceptions occur.

Answer: no

Q3: A Python decorator is applied using the '@__________' syntax.

Answer: decorator_name

Q4: The 'unittest' module in Python provides a framework for __________ testing.

Answer: unit

Q5: The 'yield' keyword is used in Python to create a __________ function.

Answer: generator

Q6: The 'filter()' function returns an iterator containing the elements for which the __________ function returns True.

Answer: filtering

Q7: The 'contextmanager' decorator simplifies the creation of __________ managers in Python.

Answer: context

Fill in the Blank 8: The 'map()' function applies a specified __________ to all items in an iterable.

Answer: function

Q9: 'DRY' stands for 'Don't __________ Yourself,' emphasizing code __________.

Answer: Repeat, reusability

Q10: A __________ copy creates a new object and recursively copies the objects found in the original.

Answer: deep