CHAPTER -6

Flow of Control

2MARK QUESTIONS

1Q: What is the purpose of control structures in programming?

Answer:

Control structures dictate the order in which statements are executed in a program, allowing for decision-making and iteration.

2Q: Differentiate between the 'if' statement and the 'else if' (elif) statement.

Answer:

The 'if' statement is used for primary decision-making. 'elif' is an abbreviation for "else if" and is used for additional conditions after the initial 'if' statement.

3 Q: Explain the concept of a loop in programming. Provide an example of a loop in Python.

Answer:

A loop is a control structure that repeats a block of code until a certain condition is met. Example in Python:

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

4Q: What is the significance of the 'break' statement in a loop?

Answer:

The 'break' statement is used to exit a loop prematurely, regardless of the loop condition.

5Q: How does the 'continue' statement differ from the 'break' statement in a loop?

Answer:

The 'continue' statement skips the rest of the loop's code and moves to the next iteration, while 'break' exits the loop entirely.

6Q: Explain the purpose of the 'switch' statement in programming.

Answer:

The 'switch' statement is used for multi-way branching, allowing the program to choose among several alternatives based on the value of an expression.

7Q: Differentiate between a 'while' loop and a 'for' loop.

Answer:

A 'while' loop continues executing as long as a given condition is true. A 'for' loop iterates over a sequence (e.g., a range of numbers) for a specified number of times.

8Q: What is the role of the 'else' clause in a 'for' or 'while' loop?

Answer:

The 'else' clause in a loop is executed when the loop condition becomes false or when the loop completes its iterations.

9Q: How does a 'do-while' loop differ from a 'while' loop?

Answer:

In a 'do-while' loop, the loop body is executed at least once before checking the loop condition, ensuring that the code inside the loop is run at least once.

10Q: Explain the concept of nested loops in programming. Provide an example.

Answer:

Nested loops refer to the situation where one loop is placed inside another. Example in Python:

python for i in range(3): for j in range(2):

print(i, j)

4MARK QUESTIONS

Q1: Explain the difference between 'break' and 'continue' statements in Python loops. Provide examples to illustrate their usage.

Answer:

The 'break' statement is used to exit a loop prematurely, stopping the loop's execution, and moving to the next statement after the loop. Example:

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

The 'continue' statement, on the other hand, skips the rest of the loop's code for the current iteration and moves to the next iteration. Example:

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

Q2: Discuss the advantages and disadvantages of using the 'switch' statement in programming. Provide an example of its usage in Python.

Answer:

Advantages:

Enhances code readability for multiple branching conditions.

Reduces the need for nested 'if-elif-else' structures.

Disadvantages:

Not natively available in Python.

Can lead to less maintainable code.

Example (using a dictionary as a workaround in Python):

```
def switch_case(case):
    switch_dict = {
        'case1': 'This is case 1',
```

```
        'case2': 'This is case 2',
        'default': 'This is the default case'
    }
    return switch_dict.get(case, 'Invalid case')
result = switch_case('case1')
```

Q3: Compare and contrast 'global' and 'local' variables in Python. Provide examples to illustrate their scopes.

Answer:

Global variables are defined outside of functions and can be accessed throughout the entire program.

Local variables are defined within a function and are only accessible within that function.

Example:

```
global_var = 10
def example_function():
    local_var = 5
    print(global_var)  # Accessing global variable
    print(local_var)   # Accessing local variable
example_function()
```

Q4: Discuss the concept of a Python generator function and its advantages over using lists. Provide an example of a generator function.

Answer:

A generator function is a special type of function that yields values one at a time, allowing iteration over large datasets without loading them entirely into memory.Advantages include memory efficiency and lazy evaluation.

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]
```

Q5: Explain the purpose of a 'docstring' in Python and provide examples demonstrating its usage.

Answer:

A 'docstring' is a string literal used to document a module, function, class, or method in Python.

It is placed as the first statement within a module, function, class, or method.

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
```

Q6: Discuss the principles of exception handling in Python. Provide examples illustrating the use of 'try', 'except', 'else', and 'finally' blocks.

Answer:

Exception handling in Python involves using 'try', 'except', 'else', and 'finally' blocks to manage errors gracefully.

Example:

```
try:
    result = 10 / 0
```

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

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

Answer:

A decorator is a function that modifies the behavior of another function in Python.

It is applied using the '@decorator_name' syntax.

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

Answer:

The 'contextlib' module in Python facilitates context management using the 'contextmanager' decorator.

It simplifies the creation of context managers.

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.")
```

Q9: Compare and contrast 'deep copy' and 'shallow copy' in Python. Provide examples illustrating each

Answer:

A 'deep copy' creates a new object and recursively copies the objects found in the original.

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)
```

Q10: Discuss the purpose and usage 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 facilitate code organization, reusability, and maintainability.

A module is a single file containing Python code. A package is a collection of modules organized in a directory hierarchy.

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
```

Feel free to use these questions for educational purposes and adapt them as needed for your specific context.

4MARK QUESTIONS

Q1: Discuss the principles of Object-Oriented Programming (OOP) and explain how they are implemented in Python. Provide examples.

Answer:

Principles of OOP include encapsulation, inheritance, and polymorphism. In Python, classes and objects are used to implement OOP.

Example:

```
class Animal:
    def __init__(self, name):
        self.name = name
 def speak(self):
        pass
class Dog(Animal):
    def speak(self):
        return "Woof!"
my_dog = Dog("Buddy")
print(my_dog.speak())
```

Q 2: Elaborate on the concepts of 'map()', 'filter()', and 'reduce()' functions in Python. Provide examples illustrating the use of each.

Answer:

'map()' applies a function to all items in an iterable.

'filter()' filters elements based on a function's result.

'reduce()' applies a function cumulatively to the items of an iterable.

Example:

```
numbers = [1, 2, 3, 4]
# map()
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
```

Q3: Describe the purpose of the 'unittest' module in Python and provide examples illustrating the creation and execution of unit tests

Answer:

'unittest' is a module for writing and running unit tests in Python.

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()
```

Q4: Explain the concept of generators in Python and provide examples illustrating the creation and usage of generators.

Answer:

Generators are special functions that yield values one at a time, saving memory.

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]
```

Q 5: Discuss the principles of code testing and the importance of writing test cases. Provide examples of writing test cases for a Python function.

Answer:

Code testing ensures reliability. Test cases validate that the code behaves as expected.

Example:

```
def add(a, b):
    return a + b
def test_add_positive_numbers():
    assert add(2, 3) == 5
```

Q6: Explore the concept of 'decorators' in Python. Provide examples demonstrating the creation and use of decorators.

Answer:

Decorators modify the behavior of other functions.

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()
```

Q7: Discuss the concepts of 'deep copy' and 'shallow copy' in Python. Provide examples illustrating each.

Answer:

'deep copy' creates a new object and recursively copies objects found in the original.

'shallow copy' creates a new object and inserts references to 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)
```

Q8: Explain the concepts of 'context managers' in Python and provide examples using the 'with' statement.

Answer:

Context managers help manage resources using 'with' statements.

Example:

```
class MyContext:
    def __enter__(self):
        print("Entering the context.")
        return self
 def __exit__(self, exc_type, exc_value, traceback):
        print("Exiting the context.")
with MyContext() as context:
    print("Inside the context.")
```

Q9: Discuss the principles of 'exception handling' in Python. Provide examples illustrating the use of 'try', 'except', 'else', and 'finally' blocks.

Answer:

Exception handling manages errors using 'try', 'except', 'else', and 'finally' blocks.

Example:

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

Q10: Describe the purpose and usage of 'docstrings' in Python. Provide examples illustrating the creation of 'docstrings' for a function and a class.

Answer:

'docstrings' are used to document code for better 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
class Person:
    """
    This class represents a person with a name and age.
    """
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Multiple-Choice Questions (MCQs):

Q 1: What is the purpose of the 'else' clause in a Python 'try-except' block?

a) To handle exceptions

b) To execute code regardless of whether an exception occurred or not

c) To define the exception type

d) To terminate the program

Answer: b) To execute code regardless of whether an exception occurred or not

Q 2: What does the 'map()' function do in Python?

a) Filters elements from a list

b) Applies a function to each item in an iterable

c) Reduces the iterable to a single value

d) Checks if elements are present in an iterable

Answer: b) Applies a function to each item in an iterable

Q 3: What is the primary purpose of a Python decorator?

a) To add comments to the code

b) To modify the behavior of another function

c) To create standalone functions

d) To define class attributes

Answer: b) To modify the behavior of another function

Q 4: What does the 'yield' keyword indicate in a Python function?

a) It returns a value from the function

b) It raises an exception

c) It defines a class method

d) It creates a generator function

Answer: **d**) It creates a generator function

Q 5: In Python, what is the purpose of the 'unittest' module?

a) To perform unit testing

b) To manipulate strings

c) To handle exceptions

d) To create decorators

Answer: a) To perform unit testing

Q 6: What does a 'shallow copy' do in Python?

a) Creates a new object and recursively copies objects found in the original

b) Creates a new object and inserts references to objects found in the original

c) Deletes objects from the original

d) Performs mathematical operations on objects

Answer: b) Creates a new object and inserts references to objects found in the original

Q 7: What does the 'with' statement facilitate in Python?

a) Exception handling

b) File handling and resource management

c) String manipulation

d) Code commenting

Answer: b) File handling and resource management

Q 8: Which function is used for filtering elements based on a given condition in Python?

a) reduce()

b) map()

c) filter()

d) zip()

Answer: c) filter()

Q 9: What is the primary purpose of the 'contextmanager' decorator in Python?

a) To manage context menus in GUIs

b) To handle exceptions in contexts

c) To simplify the creation of context managers

d) To manipulate context variables

Answer: c) To simplify the creation of context managers

Q 10: In Python, what is the role of 'docstrings'?

a) To perform mathematical operations

b) To define class attributes

c) To create decorators

d) To document code for better understanding

Answer: d) To document code for better understanding

Fill in the Blanks:

Q1: The 'break' statement is used to exit a loop _____.

Answer: prematurely

Q2: 'DRY' stands for 'Don't _____ Yourself,' emphasizing code _____.

Answer: Repeat, reusability

Q3: The 'else' clause in a 'try-except' block is executed when _____ exceptions occur.

Answer: no

Q4: A Python decorator is applied using the '@_____' syntax.

Answer: decorator_name

Q5: 'map()' applies a specified _____ to all items in an iterable.

Answer: function

Q6: The 'unittest' module in Python provides a framework for _____ testing.

Answer: unit

Q7: 'shallow copy' creates a new object and inserts _____ to objects found in the original.

Answer: references

Q8: The 'with' statement simplifies file handling and _____ management in Python.

Answer: resource

Q9: The 'contextmanager' decorator is used to simplify the creation of Python _____ managers.

Answer: context

Q10: A 'docstring' is a string literal used to document a module, function, class, or _____ in Python.

Answer: method