Unit 2: Control Statements

Introduction to Control Statements

- Control statements dictate the flow of execution in a program.
- **Types**:
  - **Selection Statements**: Make decisions and execute different code based on conditions.
  - **Iteration Statements**: Repeat code a certain number of times or while a condition is true.
  - **Jump Statements**: Break, continue, and return statements that alter the flow of control.

Selection Statements

- **if Statements:**
- Execute a block of code only if a specified condition is true.
- **Syntax:**

```
if condition:
    # code block


if:

elif:

else:
```

```
x=2
if x>0:
    print("x is positive")
else:
    print("x is not positive")


x is positive
```

Selection Statements

- **match-case Statement (Python 3.10+)**
- Simplifies multiple conditional checks by matching patterns.
- **Syntax:**

```
match variable:
    case pattern_1:
        # code block
    case pattern_2:
        # code block
    case _:
        code block
```

```
a = int(input("Enter a number: "))
match a:
    case 1:
        print("It is 1")
    case 2:
        print("It is 2")
    case _:
        print("It is neither 1 nor 2")
```

Using if-else as Ternary Operator

- A shorthand for simple if-else statements.
- Ternary conditional operator allows you to write more concise and readable conditional assignments.
- **Syntax:**
  - variable = value_if_true if condition else value_if_false

```
x = int(input("Enter a number to check if it is positive or not: "))
if x > 0:
    result = "Positive"
else:
    result = "Not positive"
print(result)
```

```
# Using if else as ternary operator
y = int(input("Enter a number again: "))
res = "Positive" if y > 0 else "Negative"
print(res)
```

Looping Statements

- **for Loops**
  - Repeat over a sequence (e.g., list, tuple, string).
  - Execute a block of code multiple times, once for each element in the sequence.
- **Syntax:**
  - for element in sequence:
  - # code block

```
for char in "Hello":
    print(char)
```

```
# Define a list of numbers
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    print(number)
```

Looping Statements

- **while Loops**
  - Repeat a block of code as long as a condition is true.
  - Execute a block of code repeatedly based on a condition.
- Syntax:
  - while condition:
  - # code block

```
count = 0
while count < 5:
    print(count)
    count+=1
```

The else Clause after for or while Loops

- The else clause executes after the loop completes normally (i.e., not terminated by a break statement).
- Execute a block of code once after the loop finishes.
- Syntax:

```
for element in sequence:
    # code block
else:
    # code block after loop
```

```
while condition:
    # code block
else:
    # code block after loop
```

```python
for num in range(5):
    print(num)
else:
    print("Loop completed")
```

The break and continue Statements

- **break Statement**
  - Terminates the loop prematurely when a certain condition is met.
  - Immediately exits the loop and continues execution after the loop.
- Syntax:

```
for item in iterable:
    if condition:
        break
```

```
for num in range(1, 6):
    if num == 4:
        break
    print(num)
else:
    print("Loop completed normally")
print("Loop terminated due to break statement")
```

The break and continue Statements

- **continue Statement**
  - Skips the rest of the current iteration and moves to the next iteration of the loop.
  - Immediately exits the loop and continues execution after the loop.
- Syntax:

```
for item in iterable:
    if condition:
        continue
```

```
for num in range(1, 6):
    if num == 3:
        continue
    print(num)
else:
    print("Loop completed normally")
print("Loop terminated")
```

The pass Statement

- Placeholder indicating no action to be taken.
- Used when syntactically required but no action is needed.
- Used in empty code blocks like in classes or functions when you plan to implement them later.
- Syntax:

  if condition:

    pass

```
for num in range(1, 6):
    if num == 3:
        pass
    else:
        print(num)
else:
    print("Loop completed normally")
print("Loop terminated")
```

Questions

- Explain the differences between for and while loops in Python. Provide examples to illustrate when each type of loop would be most appropriate to use and discuss potential advantages and disadvantages of each.
- Describe the purpose of the else clause in Python loops. How does it differ between for and while loops? Provide examples demonstrating the behavior of the else clause in both types of loops and discuss scenarios where it might be useful.
- Discuss the roles of the break and continue statements in Python loops. Explain how they differ in functionality and provide examples illustrating their usage. Describe situations where each statement would be most appropriately used and discuss potential benefits of using them.
- Explain the purpose of the pass statement in Python. Provide examples demonstrating when and why it might be used in code. Discuss its role in maintaining code structure and handling situations where no action is needed within a block of code.

Questions

- Write a Python program that uses a for loop to print all the even numbers between 1 and 10.
- Write a Python program that prompts the user to guess a number between 1 and 5. Use a while loop to allow the user to keep guessing until they guess correctly. If the user guesses correctly, print "You guessed it!" otherwise print "Try again!".
- Write a Python program that uses a for loop to iterate over a list of numbers. If a number is greater than 5, print the number and then use the break statement to exit the loop.
- Write a Python program that defines an empty function called placeholder_function. Inside the function, use the pass statement to indicate that no code should be executed yet.

Questions

- Write a Python program that simulates a basic calculator. The program should repeatedly prompt the user to enter two numbers and select an operation from a menu. The menu should include options for addition, subtraction, multiplication, and division. After performing the selected operation, the program should display the result. Implement the following features in your program:
  - Use a while loop to allow the user to perform multiple calculations until they choose to exit.
  - Use the else clause to display a message when the user decides to exit the calculator.
  - Use the break statement to exit the loop when the user chooses to exit.
  - Use the pass statement to handle cases where the user selects an invalid operation.
- Your program should provide a user-friendly interface, guiding the user through each step of the calculation process.