Python Basics Manual- Session 3

Objective

The objective of this manual is to provide a comprehensive guide on evaluating expressions, understanding conditional statements, and using for loops in programming. By the end of this session, you will be able to:

- Understand the complex data types like Sets and Dictionary.
- Understand and apply operator precedence and associativity in expressions.
- Use conditional statements (`if`, `elif`, `else`) to control the flow of your programs.
- Execute code based on different conditions.
- Understand the flow of control in conditional statements.

Table of Contents

1. Introduction to Sets and Dictionary

1.1 Sets

Sets are unordered collections of unique elements. They are useful for performing mathematical set operations like union, intersection, difference, and symmetric difference.

Creating a Set

- **Usage**: Define a set using curly braces {} or the set() function.
- **Example**:

```
my_set = {1, 2, 3, 4, 5}
another_set = set([1, 2, 3, 4, 5])
```

Set Operations

Union (|)

- **Usage**: Combines elements from both sets, removing duplicates.
- **Example**:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
result = set1 | set2  # result is {1, 2, 3, 4, 5}
```

Intersection (&)

- **Usage**: Returns elements present in both sets.
- **Example**:

```
result = set1 & set2  # result is {3}
```

Difference (-)

- **Usage**: Returns elements present in the first set but not in the second set.
- **Example**:

```
result = set1 - set2  # result is {1, 2}
```

Symmetric Difference (^)

- **Usage**: Returns elements present in either set but not in both.
- **Example**:

```
result = set1 ^ set2  # result is {1, 2, 4, 5}
```

1.2. Dictionaries

A dictionary is an unordered collection of key-value pairs. Dictionaries are defined using curly braces {}. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated.

Examples:

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
print(my_dict['a'])     # 1
my_dict['d'] = 4       # Adds a new key-value pair 'd': 4
print(my_dict)         # {'a': 1, 'b': 2, 'c': 3, 'd': 4}
del my_dict['b']       # Deletes the key 'b' and its value
print(my_dict)         # {'a': 1, 'c': 3, 'd': 4}
```

2. Introduction to Control Statements

In programming, controlling the flow of execution and performing repetitive tasks are fundamental concepts. This session will cover how to evaluate expressions with the correct precedence and associativity, how to use conditional statements to control the flow based on different conditions, and how to iterate over sequences using for loops.

2.1 Evaluating Expressions

Expressions are combinations of values and operators that Python evaluates to produce another value. Understanding how expressions are evaluated is crucial for writing correct and efficient code.

2.1.1 Operator Precedence

Operator precedence determines the order in which operators are evaluated in an expression. Operators with higher precedence are evaluated before operators with lower precedence. For example, in the expression `3 + 4 * 2`, the multiplication is performed before the addition because `*` has a higher precedence than `+`.

Table of Common Operator Precedence (Highest to Lowest):

1. Parentheses: `()`
2. Exponentiation: `**`
3. Unary plus and minus: `+x`, `-x`
4. Multiplication, Division, Floor Division, Modulus: `*`, `/`, `//`, `%`
5. Addition and Subtraction: `+`, `-`
6. Comparison: `==`, `!=`, `>`, `<`, `>=`, `<=`
7. Logical NOT: `not`
8. Logical AND: `and`
9. Logical OR: `or`

2.1.2 Operator Associativity

Operator associativity determines the order in which operators of the same precedence are evaluated. Most operators in Python are left-associative, meaning they are evaluated from left to right. For example, in the expression `5 - 3 - 2`, the operations are performed as `(5 - 3) - 2`.

2.1.3 Examples of Operator Precedence and Associativity

Example 1: Operator Precedence

```
result = 3 + 4 * 2  # result is 11 because 4 * 2 is evaluated first, then 3 + 8
```

Example 2: Operator Associativity

```
result = 5 - 3 - 2  # result is 0 because (5 - 3) is evaluated first, then 2 is subtracted
```

3. Conditional Statements

Conditional statements allow you to execute different blocks of code based on certain conditions. Python provides `if`, `elif`, and `else` statements to implement these conditionals.

3.1 Introduction to Conditional Statements

Conditional statements evaluate a boolean expression and execute the corresponding block of code if the expression is true.

3.2 The `if` Statement

The `if` statement executes a block of code if its condition is true.

Syntax:

```
if condition:
    # code to execute if condition is true
```

Example:

```
x = 10
if x > 5:
```

```
    print("x is greater than 5")
```

3.3 The `elif` Statement

The `elif` (else if) statement allows you to check multiple conditions. If the first condition is false, it checks the next condition.

Syntax:

```
if condition1:
    # code to execute if condition1 is true
elif condition2:
    # code to execute if condition2 is true
```

Example:

```
x = 10
if x > 15:
    print("x is greater than 15")
elif x > 5:
    print("x is greater than 5")
```

3.4 The `else` Statement

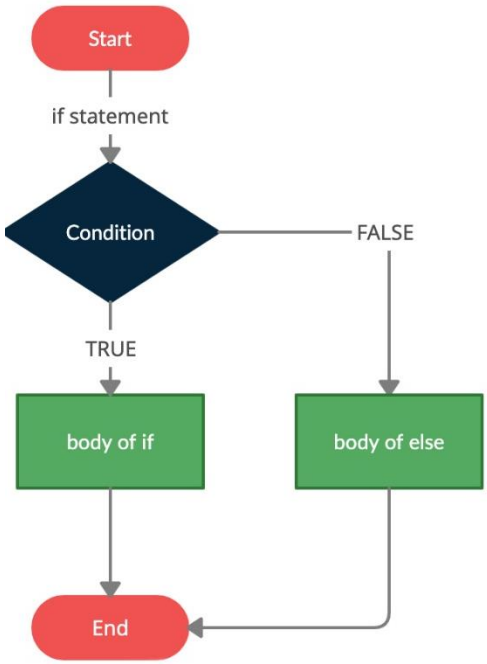
The `else` statement executes a block of code if none of the preceding conditions are true.

Syntax:

```
if condition1:
    # code to execute if condition1 is true
elif condition2:
    # code to execute if condition2 is true
else:
    # code to execute if none of the above conditions are true
```

Example:

```
x = 3
if x > 15:
    print("x is greater than 15")
elif x > 5:
    print("x is greater than 5")
else:
    print("x is 5 or less")
```



3.5 Examples of Conditional Statements

Example 1: Using `if`

```
age = 20
if age >= 18:
    print("You are an adult.")
```

Example 2: Using `if` and `elif`

```
score = 85
if score >= 90:
    print("Grade: A")
elif score >= 80:
    print("Grade: B")
```

Example 3: Using `if`, `elif`, and `else`

```
temperature = 30
if temperature > 35:
    print("It's very hot.")
elif temperature > 25:
    print("It's warm.")
else:
    print("It's cool.")
```

4. Flow of Control in Conditional Statements

Understanding the flow of control in conditional statements helps you determine the sequence in which blocks of code are executed. Python evaluates conditions from top to bottom and executes the block of code for the first true condition. If none of the conditions are true, the code inside the `else` block is executed (if it exists).

Example:

```
num = 10
if num > 20:
    print("Number is greater than 20")
elif num > 10:
    print("Number is greater than 10")
elif num > 5:
    print("Number is greater than 5")
else:
    print("Number is 5 or less")
# Output: "Number is greater than 5" (because 10 > 5 is the first true
condition)
```

5. Conclusion

This session covered the fundamentals of evaluating expressions, using conditional statements, and iterating with for loops. Mastery of these concepts will provide a solid foundation for more advanced programming topics. By understanding operator precedence and associativity, you can write expressions that evaluate as expected. By using conditional statements, you can control the flow of your programs based on different conditions. Finally, by using for loops, you can efficiently iterate over sequences to perform repetitive tasks.