Artificial Intelligence

BS (CS) _SP_2024

# Lab_01 Manual


## Learning Objectives:

1. Variables
2. Control Statements
3. List, Sets, Tuples, Dictionaries
4. Classes and Objects
5. Local and Global Variables

# Lab Manual

## Basics of Python

## Variables

In Python, variables are used to store and manage data. Each variable has a name and a value, and the type of the variable indicates what kind of data it can hold.

### Variable Naming Rules:

- Variable names in Python can contain alphanumerical characters a-z, A-Z, 0-9 and some special characters such as _.
- Normal variable names must start with a letter.
- By convention, variable names start with a lower-case letter, and Class names start with a capital letter.
- In addition, there are a number of Python keywords that cannot be used as variable names. These keywords are:

and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield

### Syntax:

```
a = 10

name = "John"

is_student = True

#Taking input from a user

name = input ("Enter student name: ")
```

## Operators

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Membership Operators

**Arithmetic Operators:**

| Operator | Example |
|---|---|
| Addition | a + b = 30 |
| Subtraction | a - b =10 |
| Multiplication | a * b |

| | |
|---|---|
| Division | a / b |
| Modulus | b % a |
| Exponent | a ** b |

### Comparison Operators:

| Operator | Description | Example |
|---|---|---|
| == | If the values of two operands are equal, then the condition becomes true | a == b |
| != | If values of two operands are not equal, then condition becomes true. | a != b |
| < > | If values of two operands are not equal, then condition becomes true. | a < > b |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. | a > b |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. | a < b |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | a >= b |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | a <= b |

### Logical Operators:

| Operator | Description | Example |
|---|---|---|
| and | If both the operands are true, then condition becomes true. | a and b |
| or | If any of the two operands are non-zero, then condition becomes true. | a or b |
| not | Used to reverse the logical state of its operand | a not b |

### Membership Operators:

| Operator | Description | Example |
|---|---|---|
| in | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | if a in b then:<br>statement |
| not in | Evaluates to true if it does not find a variable in the specified sequence and false otherwise. | if a not in b then:<br>statements; |

## Control Statements

In Python, control statements are used to guide the program's flow of operations. Depending on specific conditions, they choose which statements to perform first or repeatedly go through a list of statements.

### If-else Statement:

```
if condition:
    # Statements
elif another_condition:
```

```
    # Statements
else:
   #Statemets
```

**For Statement**

```
item_list = []
for variable in item_list:
        #Statements
```

**While Statement**

```
while condition:
    # Code to be executed as long as the condition is true
else:
```

## List

List are ordered collection of data. They are mutable (changeable) and allow duplicate elements. They are created using square brackets []. Data can be different types. Lists are versatile and can be used to store and manipulate sequences of items.

**Syntax**

```
list_A = ['a','b','c','d','BS_CS', 0343]
print(list_A)
print(list_A[1:4])      or      print(list_A[:4])      #Slicing
```

Different operations can be performed on list

| Operation | Description | Syntax |
|---|---|---|
| **Append** | Add an element at the end | list_A.append(1,2,3) |
| **Insert** | Inserts an element at a specific index | list_A.insert(1, 'Python') |
| **Extend** | Adds more than one element at the end of the list | list_A.extend(['Python', 'Java' , 'Perl']) |
| **Remove** | Removes the first item with the specified value | list_A.remove( 'Perl') |
| **Pop** | Removes last element or removes the element at the specified position. | list_A.pop();<br>list_A.pop(2);  #Remove? |
| **Index** | Returns the index of the first element with the specified value | list_A.index(0343); |
| **Sorting** | Sorts the list. And reverse() reverses the order of the list | list_A.sort();<br>list_A.reverse(); |

**Built-in Functions:**

| Function | Description |
|---|---|
| cmp (list1, list2) | Compares elements of both lists. |
| len (list1) | Gives the total length of the list. |
| max (list1) | Returns item from the list with max value. |
| min (list1) | Returns item from the list with min value. |
| list (seq) | Converts a tuple into list. |

# Set

Set is unordered collection of data. They are mutable and don't allow duplicate elements (Unique elements). They are created using curly braces {} or the set() constructor. Sets are useful for mathematical operations like union, intersection, and difference.

basket = {'Apple', 'Orange', 'pearl', 'Banana', 1, 2, 57}

Example:

Set1 = {values}  Set2 = {values}

Different operations can be performed on set

| Operation | Description | Syntax |
|---|---|---|
| **Union** | All the elements of Set1 or Set2 | new_set = Set1 \| Set2<br>Set1.union(Set2) |
| **Intersection** | Include the common elements between the two sets. | new_set = Set1 **&** Set2<br>Set1.intersection(Set2) |
| **Difference** | Elements of set1 that are not present on set2. | new_set = Set1 - Set2<br>Set1.difference(Set2) #Output {1,3,4,5,6}<br>Set2.difference(Set1) #Output? |
| **Symmetric difference** | all elements of set1 and set2 without the common elements | new_set = Set1 ^ Set2<br>Set1.symmetric_difference(Set2) |
| **Subset** | Returns **True** if another set contains this set | new_set = Set1 <= Set2<br>Set1.issubset(Set2) |
| **Superset** | Returns the index of the first element with the specified value | new_set = Set1 >= Set2<br>Set1.issuperset(Set2) |
| **Add** | Sorts the list. And reverse() reverses the order of the list | new_set.add(4) |
| **Remove** | Removes the element from a set. It returns a Key Error if element is not part of the set. | new_set.remove(2) |
| **Discard** | Removes the element from the set, and doesn't raise the error | new_set.discard(3) |
| **Clear** | Removes all elements from the set | new_set.clear() |

## Tuples

Tuples are ordered, immutable (unchangeable), and allow duplicate elements. They are created using parentheses (). Tuples are often used for fixed collections of items where immutability is desired. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

```
data = (‘Apple’, ‘Orange’, ’pearl’, ‘Banana’, 1, 2, 57)

new_tuple = data [1:4]                    #create a new tuple by extracting a portion of an existing tuple using slicing.
```

## Dictionary

A dictionary is a collection which is unordered, changeable, and indexed. In Python dictionaries

are written with curly brackets, and they have keys and values.

**Syntax:**

```
data = { key : value }
```

**Example:**

```
thisdict = { "brand": "Ford", "model": "Mustang", "year": 1964}

print(thisdict)

# Accessing Items

x = thisdict["model"]          or          x = thisdict.get("model")
```

## Classes

A class is a blueprint for creating objects. It serves as a template that defines the attributes and behaviors of an object.

It encapsulates data (attributes) and methods (functions) that operate on that data.

Also, promote code reusability and maintainability.

**Syntax:**

```
class ClassName:
        #class attributes and methods
        def functionName:
```

## Objects

Objects are instances of a class, created using the class name followed by parentheses.

Objects encapsulate the attributes and methods defined in the class.

### Syntax:

object_name = ClassName()

my_car = Car("Toyota", "Camry", 2022)

## Local and Global Variables

### Outside the class

```
# Global variables for Employee class
variable1 = "value1"
variable2 = value2


class ClassName:
    def __init__(self, variable1, variable2):
        self.variable1 = variable1
        self.variable2 = variable2
    def display():
        print(f" Variable 1: {variable1},\nVariable 2: {variable2}”)
```

### Inside the class

```
class ClassName:
variable1 = "value1"
variable2 = value2
    def __init__(self, variable1, variable2):
        self.variable1 = variable1
        self.variable2 = variable2
```

```
def display():
    print(f" Variable 1: {variable1},\nVariable 2: {variable2}”)
```

### Inside the function

```
def some_function():
    local_variable = 12
    # local_variable is a local variable
```

## Lab Task

1. Write a Python program that reads a positive integer and convert it into a Roman numeral.
2. Write a program that takes input for a list of students along with their grades in different subjects (like ICT, OOP, Data Structure and Database). Use control statements to calculate the average grade, identify the highest and lowest grades, and categorize students based on their performance. Use a tuple to store student information and grades.
3. Create a class called BankAccount which represents a bank account, having as attributes: accountNumber, name, balance. Create a constructor with parameters: accountNumber, name, balance. Create a Deposit( ) method which manages the deposit actions. Create a withdrawal( ) method which manages the withdrawal actions. Create a bankFees( ) method to apply the bank fees with a percentage of 5% of the balance account. Create a display( ) method to display account details.
4. Create a Time class and initialize it with hours and minutes.
    a) Make a method addTime which should take two time object and add them. E.g. (2h and 50min) + (1h and 20min) is (4h and 10min).
    b) Make a method displayTime which should print the time.
    c) Make a method DisplayMinutes which should display the total minutes in time. E.g. (1h and 2 min) should display 62 minutes.