PERIYAR UNIVERSITY

NAAC 'A++' Grade - State University - NIRF Rank 56 – State Public University Rank 25
SALEM - 636 011, Tamil Nadu, India.

CENTRE FOR DISTANCE AND ONLINE EDUCATION (CDOE)

BACHELOR OF COMPUTER SCIENCE

SEMESTER - I

COURSE: PYTHON PROGRAMMING

(Candidates admitted from 2024 onwards)

PERIYAR UNIVERSITY

CENTRE FOR DISTANCE AND ONLINE EDUCATION (CDOE)

B.Sc., Computer Science 2024 admission onwards

CORE – I

Python Programming

Prepared by:

Centre for Distance and Online Education (CDOE)

Periyar University, Salem – 11.

SYLLABUS

PYTHON PROGRAMMING

Unit I : Basics of Python Programming: History of Python-Features of Python-Literal -Constants-Variables - Identifiers–Keywords-Built-in Data Types-Output Statements – Input Statements-Comments – Indentation- Operators-Expressions -Type conversions. Python Arrays: Defining and Processing Arrays – Array methods.

Unit II: Control Statements: Selection/Conditional Branching statements: if, if-else, nested if and if-elif-else statements. Iterative Statements: while loop, for loop, else suite in loop and nested loops. Jump Statements: break, continue and pass statements.

Unit III: Functions: Function Definition – Function Call – Variable Scope and its Lifetime-Return Statement. Function Arguments: Required Arguments, Keyword Arguments, Default Arguments and Variable Length Arguments- Recursion. Python Strings: String operations- Immutable Strings - Built-in String Methods and Functions - String Comparison. Modules: import statement- The Python module – dir() function – Modules and Namespace – Defining our own modules

Unit IV : Lists: Creating a list -Access values in List-Updating values in Lists-Nested lists - Basic list operations- List Methods. Tuples: Creating, Accessing, Updating and Deleting Elements in a tuple – Nested tuples– Difference between lists and tuples. Dictionaries: Creating, Accessing, Updating and Deleting Elements in a Dictionary – Dictionary Functions and Methods - Difference between Lists and Dictionaries.

Unit V : Python File Handling: Types of files in Python - Opening and Closing files- Reading and Writing files: write() and writelines() methods- append() method – read() and readlines() methods – with keyword – Splitting words – File methods - File Positions- Renaming and deleting files

TABLE OF CONTENTS

PYTHON PROGRAMMING

UNIT 1 – PYTHON PROGRAMMING

Basics of Python Programming: History of Python-Features of Python-Literal -Constants-Variables - Identifiers–Keywords-Built-in Data Types-Output Statements – Input Statements-Comments – Indentation-Operators-Expressions -Type conversions. Python Arrays: Defining and Processing Arrays – Array methods.

Basics of Python Programming and Array

Unit Objectives

In this unit, leaners will have a brief understanding of the history and features of python programming. The basic concepts that are need for any programming languages like constants, literals, keywords and how to declare a variable in python will be discussed. Along with the basic data types that are supported in python programming. The way how to use the input, output statement and how data type conversion of variables in done in python. Finally the unit ends with how to create and process Arrays and the methods used for array in python.

SECTION 1.1: BASICS OF PYTHON

1.1.1 – History of Python

Python was first developed by Guido van Rossum in the late 80"s and early 90"s at the National Research Institute for Mathematics and Computer Science in the Netherlands. It has been derived from many languages such as ABC, Modula-3, C, C++, Algol68, SmallTalk, UNIX shell and other scripting languages.Since early 90"s Python has been improved tremendously. Its version 1.0 was released in 1991, which

introduced several new functional programming tools. While version 2.0included list comprehension was released in 2000 by the Be Open Python Labs team.

Python 2.7 which is still used today will be supported till 2020. Currently Python 3.6.4 is already available. The newer versions have better features like flexible string representation etc., Although Python is copyrighted, its source code is available under GNU General Public License (GPL) like that Perl. Python is currently maintained by a core development team at the institute which is directed by Guido Van Rossum. These days, from data to web development, Python has emerged as very powerful and popular language. It would be surprising to know that python is actually older than Java, R and JavaScript.

1.1.2 – Features of Python

- **Simple**: Reading a program written in Python feels almost like reading english. The main strength of Python which allows programmer to concentrate on the solution to the problem rather than language itself.
- **Easy to Learn:** Python program is clearly defined and easily readable. The structure of the program is simple. It uses few keywords and clearly defined syntax.
- **Versatile:** Python supports development of wide range of applications such as simple text processing, WWW browsers and games etc..
- **Free and Open Source:** It is a Open Source Software. So, anyone can freely distribute it, read the source code, edit it, and even use the code to write new (free) programs.
- **High-level Language:** While writing programs in Python we do not worry about the low-level details like managing memory used by the program.
- **Interactive:** Programs in Python work in interactive mode which allows interactive testing and debugging of pieces of code. Programmer can easily interact with the interpreter directly at the python prompt to write their programs.
- **Portable:** It is a portable language and hence the programs behave the same on wide variety of hardware platforms with different operating systems.

- **Object Oriented:** Python supports object-oriented as well as procedure-oriented style of programming .While object-oriented technique encapsulates data and functionality with in objects, Procedure oriented at other hand, builds programs around procedure or functions.
- **Interpreted:** Python is processed at runtime by interpreter. So, there is no need to compile a program before executing it. You can simply run the program. Basically python converts source program into intermediate form called byte code.
- **Dynamic and strongly typed language:** Python is strongly typed as the interpreter keeps track of all variables types. It's also very dynamic as it rarely uses what it knows to limit variable usage.
- **Extensible:** Since Python is open source software, anyone can add low-level modules to the python interpreter. These modules enable programmers to add to or customize their tools to work more efficiently.
- **Embeddable:** Programmers can embed Python within their C, C++, COM, ActiveX, CORBA and Java Programs to give „scripting „capability for users.
- **Extensive Libraries:** Python has huge set of libraries that is easily portable across different platforms with different operating systems.
- **Easy maintenance:** Code Written in Python is easy to maintain.
- **Secure:** This Programming language is secure for tampering. Modules can be distributed to prevent altering of source code. Additionally, Security checks can be easily added to implement additional security features.
- **Robust:** Python Programmers cannot manipulate memory directly, errors are raised as exceptions that can be catch and handled by the program code. For every syntactical mistake, a simple and easy to interpret message is displayed. All these make python robust.
- **Multi-threaded:** Python supports executing more than one process of a program simultaneously with the help of Multi Threading.
- **Garbage Collection:** The Python run-time environment handles garbage collection of all python objects. For this, a reference counter is maintained to assure that no object that is currently in use is deleted.

1.1.3 – Literal Constants

Python literals or constants are the notation for representing a fixed value in source code. In contrast to variables, literals (123, 4.3, "Hello") are static values or you can say constants which do not change throughout the operation of the program or application.

Example

x=10

Here 10 is a literal as numeric value representing 10, which is directly stored in memory. However,

y=x*2

Here, even if the expression evaluates to 20, it is not literally included in source code. You can also declare an int object with built-in int() function. However, this is also an indirect way of instantiation and not with literal.

x=(int)10

Number

The value of a literal constant can be used directly in programs. For example, 7, 3.9, 'A', and "Hello" are literal constants. Numbers refers to a numeric value. You can use four types of numbers in Python program- integers, long integers, floating point and complex numbers. Numbers like 5 or other whole numbers are referred to as integers. Bigger whole numbers are called long integers.

For example, 535633629843L is a long integer. Numbers like are 3.23 and *91.5E-2 are termed as* **floating point numbers**. *Numbers of a + bj form (like -3 + 7j) are* **complex numbers**.

>>>50+40-35 55	>>>12 * 10 120	>>>96/12 8.0	>>>(-30 * 4) +500 380
>>>78/5 15	>>>78%5 3	>>>152.78//3.0 50.0	>>>152.78%3.0 2.780000000001

A Literals Boolean type can have one of the two values- True or False.

>>>bvar=True >>>print(bvar) True	>>>20==30 False	>>>”Python” True
>>>20 != 20 False	>>>”Python” != “Python3.3” True	>>>30>50 False
>>>90<=90 True	>>>87==87.0 False	>>>87>87.0 false
>>> 87<87.0 False	>>>87>=87.0 True	>>>87<=87.0 True

String

A string is a group of characters. Using Single Quotes ('): For example, a string can be written as 'HELLO'. Using Double Quotes ("): Strings in double quotes are exactly same as those in single quotes. Therefore, 'HELLO' is same as "HELLO". Using Triple Quotes (''' '''): You can specify multi-line strings using triple quotes. You can use as many single quotes and double quotes as you want in a string within triple quotes.

>>>”Hello” ‘Hello’	>>>’Hello’ ‘Hello’	>>>’’’Hello’’’ ‘Hello’

Unicode Strings

Unicode is a standard way of writing international text. That is,if you want to write some text in your native language like hindi,then you need to have a Unicode-enable text editor. Python allows you to specify Unicode text by prefixing the string with a u or U. For Example: u”Sample Unicode string” Note :The „U‟ prefix specifies that the file contains text written in language other than English

Escape Sequences

Some characters (like ", \) cannot be directly included in a string. Such characters must be escaped by placing a backslash before them.

```
>>> print("The boy replies, \"My name is Aaditya.\"")
The boy replies, "My name is Aaditya."
```

Escape Sequence	Purpose	Example	Output
\\	Prints Backslash	print("\\")	\
\'	Prints single-quote	print("\'")	'
\"	Prints double-quote	print("\"")	"
\a	Rings bell	print("\a")	Bell rings
\f	Prints form feed character	print("Hello\fWorld")	Hello World
\n	Prints newline character	print("Hello\nWorld")	Hello World
\t	Prints a tab	print("Hello\tWorld")	Hello World
\o	Prints octal value	print("\o56")	.
\x	Prints hex value	print("\x87")	+

Raw Strings

If you want to specify a string that should not handle any escape sequences• and want to display exactly as specified then you need to specify that string as a raw string. A raw string is specified by prefixing r or R to the string.

Example

>>>print(R ”What\’s is your name?”)

what\’s is your name?

1.1.4 – Variables

In programming, a variable is a container (storage area) to hold data. Identifiers are things like variables. An Identifier is utilized to recognize the literals utilized in the program. The standards to name an identifier are given underneath.

- The variable's first character must be an underscore or alphabet (_).
- Every one of the characters with the exception of the main person might be a letter set of lower-case(a-z), capitalized (A-Z), highlight, or digit (0-9).
- White space and special characters (!, @, #, %, etc.) are not allowed in the identifier name. ^, &, *).
- Identifier name should not be like any watchword characterized in the language.

- Names of identifiers are case-sensitive; for instance, my name, and MyName isn't something very similar.

Examples of valid identifiers: a123, _n, n_9, etc.

Examples of invalid identifiers: 1a, n%4, n 9, etc.

Declaring Variable and Assigning Values

Python doesn't tie us to pronounce a variable prior to involving it in the application. It permits us to make a variable at the necessary time. In Python, we don't have to explicitly declare variables. The variable is declared automatically whenever a value is added to it. The equal (=) operator is utilized to assign worth to a variable.

Example

```
number = 10
site_name = 'programiz.pro'
print(site_name)
```

Output

```
'programiz.pro'
```

Example: Assigning multiple values to multiple variables

```
a, b, c = 5, 3.2, 'Hello'

x=y=z=50

print (a)  # prints 5

print (b)  # prints 3.2

print (c)  # prints Hello
```

Local Variable

The variables that are declared within the function and have scope within the function are known as local variables. Let's examine the following illustration.

Example -

```
def add():      # Defining local variables. They has scope only within a function

   a = 20

   b = 30

   c = a + b

   print("The sum is:", c)

add()  # Calling a function
```

Output:

The sum is: 50

Global Variables

Global variables can be utilized all through the program, and its extension is in the whole program. Global variables can be used inside or outside the function. By default, a variable declared outside of the function serves as the global variable. Python gives the worldwide catchphrase to utilize worldwide variable inside the capability. The function treats it as a local variable if we don't use the global keyword.

Example -

```
x = 101  # Declare a variable and initialize it
def mainFunction():  # Global variable in function
   # printing a global variable
   global x
   print(x)
   # modifying a global variable
   x = 'Welcome To Python’
   print(x)
mainFunction()
print(x)
```

Output:

101

Welcome To Python

Welcome To Python

1.1.5 – Identifiers

Identifiers (also referred to as *names*) are described by the following lexical definitions. A Python identifier is the name given to a variable, function, class, module or other object. An identifier can begin with an alphabet (A – Z or a – z), or an underscore (_) and can include any number of letters, digits, or underscores. Spaces are not allowed. Python will not accept @, $ and % as identifiers. Furthermore, Python is a case-sensitive language. Thus, Hello and hello both are different identifiers. In Python, a class name will always start with a capital letter. Examples of Valid and Invalid Names for Creating Identifiers are :

Valid

MyName
My_Name

Your_Name

Invalid

My Name (Space is not allowed)

3dfig (cannot start with a digit)

Your#Name (Only alphabetic character, Underscore (_) and numeric are allowed

1.1.6 – Keywords

Python has a list of reserved words known as keywords. Every keyword has a specific purpose and use. In the upcoming chapters, we will look into the use of these keywords in programming. A list of reserved keywords in Python are

and	del	from	None	True
as	elif	global	nonlocal	try

assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	
def	for	lambda	return	

1.1.7 – Built –in data types

The data stored in the memory can be of many types. For example, a person's name is stored as an alphabetic value and its address is stored as an alphanumeric value. Sometimes, we also need to store answer in terms of only 'yes' or 'no', i.e., true or false. This type of data is known as Boolean data.

Python has six basic data types which are as follows:

1. Numeric
2. String
3. List
4. Tuple
5. Dictionary
6. Boolean

Numeric

Numeric data can be broadly divided into integers and real numbers (i.e., fractional numbers). Integers can themselves be positive or negative. Unlike many other programming languages, Python does not have any upper bound on the size of integers. The real numbers or fractional numbers are called floating point numbers in programming languages. Such floating point numbers contain a decimal and a fractional part.

Example

```
>>> num1=2
>>>num2=2.5
>>>num1
 2  # Output
```

```
>>>num2
2.5  # Output
```

Note : In all the earlier versions of Python 3, slash (/) operator worked differently. When both numerator and denominator are integers, then the result will be an integer. The slash operator removes the fraction part.

The result becomes a floating number when either the numerator or the denominator is a floating number. When both the numerator and the denominator are floating numbers, the result is again a floating number. The division operator provides accurate results even when both the numerator and the denominator are integers.

Example

```
>>> 5/2
1.5      # Output
```

String

Besides numbers, strings are another important data type. Single quotes or double quotes are used to represent strings. A string in Python can be a series or a sequence of alphabets, numerals and special characters. Similar to C, the first character of a string has an index 0.

There are many operations that can be performed on a string. There are several operators such as slice operator ([]) and [:]), concatenation operator (+), repetition operator (*), etc. Slicing is used to take out a subset of the string, concatenation is used to combine two or more than two strings and repetition is used to repeat the same string several times.

Example

```
>>>sample_string =”Hello”  # store string value
>>>sample_string          # display string value
 ‘Hello’                  # Output
>>>sample_string + “World” # use of + operator
‘HelloWorld’              # Output
>>>sample_string * 3       # use of * operator
‘HelloHelloHello’         # Output
```

Python also provides slice operators ([] and [:]) to extract substring from the string. In Python, the indexing of the characters starts from 0; therefore, the index value of the first character is 0.

Syntax

sample_string[start : end <:step>] #step is optional

Example

```
>>>sample_string=”Hello”
>>>sample_string[1]        # display 1st index element.
‘e’                        # Output
>>>sample_string[0:2]      # display 0 to 1st  index elements
‘He’                       # Output
>>>sample_string = "HelloWorld"
>>>sample_string[1:8:2]    # display all the alternate characters between index 1 to 8. ie, 1,3,5,7
 'elWr'                    # Output
```

List

List is the most used data type in Python. A list can contain the same type of items. Alternatively, a listcan also contain different types of items. A list is an ordered and indexable sequence. To declare a list in Python, we need to separate the items using commas and enclose them within square brackets([]). The list is somewhat similar to the array in C language. However, an array can contain only the same type of items while a list can contain different types of items.

Similar to the string data type, the list also has plus (+), asterisk (*) and slicing [:] operators for concatenation, repetition and sub-list, respectively.

Example

```
>>>first=[1,”two”,3.0,”four”]          # 1 list
>>>second=[“five”, 6]                  # 2 stnd list
>>>first                       # display 1
[1, ‘two’, 3.0, ‘four’]       # Output
>>>first+secondst list                 # concatenate 1and 2 list
 [1, ‘two’, 3.0, ‘four’, ‘five’, 6]   # Output
>>>second * 3                          # repeat 2nd list
```

```
['five', 6, 'five', 6, 'five', 6]   # Output
>>>first[0:2]                      # display sublist
[1, 'two']
```

Tuple

Similar to a list, a tuple is also used to store sequence of items. Like a list, a tuple consists of items separated by commas. However, tuples are enclosed within parentheses rather than within square brackets.

Example

```
>>>third=(7, "eight",9, 10.0)
>>>third
(7, 'eight', 9, 10.0)          # Output
```

Lists and tuples have the following differences:

- In lists, items are enclosed within square brackets [], whereas in tuples, items are enclosed within parentheses ().
- Lists are mutable whereas Tuples are immutable. Tuples are read only lists. Once the items are stored, the tuple cannot be modified.

Dictionary

It is the same as the hash table type. The order of elements in a dictionary is undefined. But, we can iterate over the following:

1. The keys
2. The values
3. The items (key-value pairs) in a dictionary

A Python dictionary is an unordered collection of key-value pairs. When we have the large amount of data, the dictionary data type is used. Keys and values can be of any type in a dictionary. Items in dictionary are enclosed in the curly-braces{} and separated by the comma (,). A colon (:) is used o separate key from value. A key inside the square bracket [] is used for accessing the dictionary items.

Example

```
>>> dict1 = {1:"first line", "second":2}  # declare dictionary
>>>dict1[3] = "third line"                  # add new item
```

```
>>> dict1                                  # display dictionary
 {1: 'first line', 'second': 2, 3: 'third line'}  #Output
>>>dict1.keys()                           # display dictionary keys
 [1, 'second', 3]                        # Output
>>>dict1.values()                         # display dictionary values
['first line', 2, 'third line']          # Output
```

Boolean

In a programming language, mostly data is stored in the form of alphanumeric but sometimes we need to store the data in the form of 'Yes' or 'No'. In terms of programming language, Yes is similar to True and No is similar to False. This True and False data is known as Boolean Data and the data types which stores this Boolean data are known as Boolean Data Types.

Example

```
>>> a = True
>>>type(a)
<type 'bool'>
>>> x = False
>>>type(x)
<type 'bool'>
```

1.1.8 – Output Statements

Python print() function prints the message to the screen or any other standard output device. In this article, we will cover about print() function in Python as well as it's various operations.

Syntax

print(value(s), sep= ' ', end = '\n', file=file, flush=flush)

Parameters:

- **value(s):** Any value, and as many as you like. Will be converted to a string before printed

- **sep='separator' :** (Optional) Specify how to separate the objects, if there is more than one.Default :' '
- **end='end':** (Optional) Specify what to print at the end.Default : '\n'
- **file :** (Optional) An object with a write method. Default :sys.stdout
- **flush :** (Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

Return Type: It returns output to the screen.

Example

```
name = "John"
age = 30
print("Name:", name)
print("Age:", age)
```

Output:

```
Name: John
Age: 30
```

How print() works in Python?

You can pass variables, strings, numbers, or other data types as one or more parameters when using the print() function. Then, these parameters are represented as strings by their respective str() functions. To create a single output string, the transformed strings are concatenated with spaces between them. In this code, we are passing two parameters name and age to the print function

Example

```
name = "Alice"
age = 25
print("Hello, my name is", name, "and I am", age, "years old.")
```

Output

```
Hello, my name is Alice and I am 25 years old.
```

String literals in Python's print statement are primarily used to format or design how a specific string appears when printed using the print() function.

- **\n:** This string literal is used to add a new blank line while printing a statement.
- **“”:** An empty quote (“”) is used to print an empty line.

```
>>>print("Python Programming \n is OOP.")
Python Programming
is OOP.
```

The **end keyword** is used to specify the content that is to be printed at the end of the execution of the print() function. By default, it is set to “\n”, which leads to the change of line after the execution of print() statement.

Example

```
#using print() with end and without end parameters.
# This line will automatically add a new line before the next print statement
print ("Python Programming is OOP ")
# This print() function ends with "**" as set in the end argument.
print ("Python Programming is OOP ", end= "&&")
print("Welcome to Python World")
```

Output:

```
Python Programming is OOP
Python Programming is OOP&&Welcome to Python World
```

Print concatenation

```
>>>print('Python Programming supports ' + OOP.')
Python Programming supports OOP
```

Print using formatting

```
>>>a,b,=10,1000
print('The value of a is {} and b is {}'.format(a,b))
The value of a is 10 and b is 1000
```

“sep” parameter in print()

The print() function can accept any number of positional arguments. To separate these positional arguments, the keyword argument “sep” is used.

Example

```
a=12
b=12
c=2024
print(a,b,c,sep="-")
```

Output:

```
12-12-2024
```

1.1.9 – Input Statements

In a programming language, the input from keyboard by user plays the most important role in executing a program. There is hardly any program which executes without some input. The input in many programs is prompted by the user and the user uses the keyboard in order to provide input for the program to execute. Python programming language also provides the facility to user to provide input from keyboard.

Prompting the input from user in Python is through function has an optional parameter, which is the prompt string. When the input() input() function. input() function is called, in order to take input from the user then the execution of program halts and waits for the user to provide an input. The input is given by the user through keyboard and it is ended by the return key. input() function interprets the input provided by the user, i.e. if user provides an integer value as input then the input function will return this integer value. On the other hand, if the user has input a String, then the function will return a string.

Example

```
>>>name = input(“What is your Name?”)
>>> print (“Hello “ + name + “!”)
 What is your Name? ‘John’
```

```
Hello John!
>>>age = input("Enter your age? ")
>>> print age
 Enter your age? 32
32                              #Output
>>>hobbies = input("What are your hobbies? ")
>>>print hobby
What are your hobbies? ['playing', 'travelling'] #Output
 ['playing', 'travelling']
>>>type(name)
<type 'str'>                    #Output
```

1.1.10 – Comments

Just like other programming languages, Python allows you to add comments in the code. Comments are used by the programmer to explain the piece of code to others as well as to himself in a simple language. Every programming language makes use of some special character for commenting, so does Python.

Python uses the hash character (#) for comments. Putting # before a text ensures that the text will not be parsed by the interpreter. Comments do not affect the programming part and the Python interpreter does not display any error message for comments. Comments show up as it is in the programming. It is a good practice to use comments for program documentation in your program so that it becomes easier for other programmers to maintain or enhance the program when required.

Example : Commenting without the use of Hash mark (#)

```
>>> 8+9 addition
```

SyntaxError: invalid syntax # Output

In the above example, 'addition' is written without the Hash mark. As a result, the interpreter accepts the word 'addition' as part of programming. Since 'addition' is not a command in Python, an error message is displayed.

Example : Commenting using Hash mark (#)

```
>>> 8+9 #addition
 17  # Output
```

Now, in this example, 'addition' is written with a Hash mark. Hence, the interpreter understands it as a comment and does not display any error message.

1.1.11 – Indentation

Python relies on indentation to define the structure of code blocks, making it unique among programming languages. Unlike languages that use explicit symbols or braces to denote code blocks, Python uses whitespace, specifically indentation, to determine the scope of statements.

In Python, indentation refers to the spacing at the beginning of a line of code that determines its grouping and hierarchy within the program's structure. Unlike many programming languages that use braces ({}) or other explicit symbols to denote code blocks, Python uses indentation to signify the beginning and end of blocks of code.

Rules of Indentation in Python

- Python's default indentation spaces are four spaces. The number of spaces, however, is entirely up to the user. However, a minimum of one space is required to indent a statement.
- Indentation is not permitted on the first line of Python code.
- Python requires indentation to define statement blocks.
- A block of code must have a consistent number of spaces.

- To indent in Python, whitespaces are preferred over tabs. Also, use either whitespace or tabs to indent; mixing tabs and whitespaces in indentation can result in incorrect indentation errors.

Example

```
site = 'prepbytes'

if site == 'prepbytes':

print('Logging on to prepbytes...')

else:

print('retype the URL.')

print('All set !')
```

Output

```
Logging on to prepbytes...

All set !
```

Example 2

```
j = 1

while(j<= 5):

print(j)

   j = j + 1
```

Output

```
1

2

3

4

5
```

1.1.12 – Operators

Operators are used to perform operations on variables and values. Python supports following operators -

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic Opertors

Arithmetic operators used between two operands for a particular operation. There are many arithmetic operators. It includes the exponent (**) operator as well as the + (addition), - (subtraction), * (multiplication), / (divide), % (reminder), and // (floor division) operators.

Consider the following table for a detailed explanation of arithmetic operators.

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 10, b = 10 =>a+b = 20
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 5 => a - b = 15

/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 4 => a * b = 80
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 =>a%b = 0
**** (Exponent)**	As it calculates the first operand's power to the second operand, it is an exponent operator.
// (Floor division)	It provides the quotient's floor value, which is obtained by dividing the two operands.

Example

```
a = 32    # Initialize the value of a
b = 6     # Initialize the value of b
print('Addition of two numbers:',a+b)
print('Subtraction of two numbers:',a-b)
print('Multiplication of two numbers:',a*b)
print('Division of two numbers:',a/b)
print('Reminder of two numbers:',a%b)
print('Exponent of two numbers:',a**b)
print('Floor division of two numbers:',a//b)
```

Output:

Addition of two numbers: 38

Subtraction of two numbers: 26

Multiplication of two numbers: 192

Division of two numbers: 5.333333333333333

Reminder of two numbers: 2

Exponent of two numbers: 1073741824

Floor division of two numbers: 5

Comparison Operators

Comparison operators mainly use for comparison purposes. Comparison operators compare the values of the two operands and return a true or false Boolean value in accordance. The example of comparison operators are ==, !=, <=, >=, >, <. In the below table, we explain the works of the operators.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal, then the condition becomes true.
<=	The condition is met if the first operand is smaller than or equal to the second operand.
>=	The condition is met if the first operand is greater than or equal to the second operand.
>	If the first operand is greater than the second operand, then the condition becomes true.

<	If the first operand is less than the second operand, then the condition becomes true.

Example:

```
a = 32      # Initialize the value of a
b = 6       # Initialize the value of b
print('Two numbers are equal or not:',a==b)
print('Two numbers are not equal or not:',a!=b)
print('a is less than or equal to b:',a<=b)
print('a is greater than or equal to b:',a>=b)
print('a is greater b:',a>b)
print('a is less than b:',a<b)
```

Output:

```
Two numbers are equal or not: False
Two numbers are not equal or not: True
a is less than or equal to b: False
a is greater than or equal to b: True
a is greater b: True
a is less than b: False
```

Assignment Operators

Using the assignment operators, the right expression's value is assigned to the left operand. There are some examples of assignment operators like =, +=, -=, *=, %=, **=, //=. In the below table, we explain the works of the operators.

Operator	Description
=	It assigns the value of the right expression to the left operand.

+=	By multiplying the value of the right operand by the value of the left operand, the left operand receives a changed value. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assigns the reminder back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

Example:

```
a = 32        # Initialize the value of a
b = 6         # Initialize the value of b
print('a=b:', a==b)
print('a+=b:', a+b)
print('a-=b:', a-b)
```

```
print('a*=b:', a*b)

print('a%=b:', a%b)

print('a**=b:', a**b)

print('a//=b:', a//b)
```

Output:

```
a=b: False

a+=b: 38

a-=b: 26

a*=b: 192

a%=b: 2

a**=b: 1073741824

a//=b: 5
```

Logical Operators

The assessment of expressions to make decisions typically uses logical operators. The examples of logical operators are and, or, and not. In the case of logical AND, if the first one is 0, it does not depend upon the second one. In the case of logical OR, if the first one is 1, it does not depend on the second one. Python supports the following logical operators. In the below table, we explain the works of the logical operators.

Operator	Description
and	The condition will also be true if the expression is true. If the two expressions a and b are the same, then a and b must both be true.

or	The condition will be true if one of the phrases is true. If a and b are the two expressions, then an or b must be true if and is true and b is false.
not	If an expression **a** is true, then not (a) will be false and vice versa.

Example:

```
a = 5          # initialize the value of a
print(Is this statement true?:',a > 3 and a < 5)
print('Any one statement is true?:',a > 3 or a < 5)
print('Each statement is true then return False and vice-
versa:',(not(a > 3 and a < 5)))
```

Output:

```
Is this statement true?: False
Any one statement is true?: True
Each statement is true then return False and vice-versa: True
```

Bitwise Operators

The two operands' values are processed bit by bit by the bitwise operators. The examples of Bitwise operators are bitwise OR (|), bitwise AND (&), bitwise XOR (^), negation (~), Left shift (<<), and Right shift (>>).

Example

```
if a = 7
   b = 6
then, binary (a) = 0111
    binary (b) = 0110

hence, a & b = 0011
     a | b = 0111
          a ^ b = 0100
```

~ a = 1000

Let, Binary of x = 0101

Binary of y = 1000

Bitwise OR = 1101

8 4 2 1

1 1 0 1 = 8 + 4 + 1 = 13

Bitwise AND = 0000

0000 = 0

Bitwise XOR = 1101
8 4 2 1

1 1 0 1 = 8 + 4 + 1 = 13

Negation of x = ~x = (-x) - 1 = (-5) - 1 = -6

~x = -6

1.

Operator	Description
& (binary and)	A 1 is copied to the result if both bits in two operands at the same location are 1. If not, 0 is copied.
| (binary or)	The resulting bit will be 0 if both the bits are zero; otherwise, the resulting bit will be 1.
^ (binary xor)	If the two bits are different, the outcome bit will be 1, else it will be 0.
~ (negation)	The operand's bits are calculated as their negations, so if one bit is 0, the next bit will be 1, and vice versa.
<< (left shift)	The number of bits in the right operand is multiplied by the leftward shift of the value of the left operand.
>> (right shift)	The left operand is moved right by the number of bits present in the right operand.

Example:

```
a = 5        # initialize the value of a
b = 6        # initialize the value of b
print('a&b:', a&b)
print('a|b:', a|b)
print('a^b:', a^b)
print('~a:', ~a)
print('a<<b:', a<<b)
print('a>>b:', a>>b)
```

Output:

```
a&b: 4
a|b: 7
a^b: 3
~a: -6
a<>b: 0
```

Membership Operators

The membership of a value inside a Python data structure can be verified using Python membership operators. The result is true if the value is in the data structure; otherwise, it returns false.

Operator	Description
in	If the first operand cannot be found in the second operand, it is evaluated to be true (list, tuple, or dictionary).
not in	If the first operand is not present in the second operand, the evaluation is true (list, tuple, or dictionary).

Example

```
x = ["Rose", "Lotus"]
```

```
print(' Is value Present?', "Rose" in x)
print(' Is value not Present?', "Riya" not in x)
```

Output:

```
Is value Present? True
Is value not Present? True
```

Identity Operators

These operators are used to check whether both operands are same or not. Suppose, x stores a value 20 and y stores a value 40. Then x is y return false and x not is y returns true.

Operator	Description
is	If the references on both sides point to the same object, it is determined to be true.
is not	If the references on both sides do not point at the same object, it is determined to be true.

Example

```
a = ["Rose", "Lotus"]
b = ["Rose", "Lotus"]
c = a
print(a is c)
print(a is not c)
print(a is b)
print(a is not b)
print(a == b)
print(a != b)
```

Output:

True

False

False

True

True

False

Operator Precedence

The order in which the operators are examined is crucial to understand since it tells us which operator needs to be considered first. Below is a list of the Python operators' precedence tables.

Operator	Description
**	Overall other operators employed in the expression, the exponent operator is given precedence.
~ + -	the minus, unary plus, and negation.
* / % //	the division of the floor, the modules, the division, and the multiplication.
+ -	Binary plus, and minus
>><<	Left shift. and right shift
&	Binary and.
^ |	Binary xor, and or
<= <>>=	Comparison operators (less than, less than equal to, greater than, greater then equal to).

<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

1.1.13 – Expressions

An expression is any legal combination of symbols (like variables, constants and operators) that represents a value. In Python, an expression must have at least one operand (variable or constant) and can have one or more operators. On evaluating an expression, we get a value. Operand is the value on which operator is applied.

Generally Expressions are divided into the following types

1. **Constant Expressions**: One that involves only constants.

Example: 8 + 9 – 2

2. **Integral Expressions**: One that produces an integer result after evaluating the expression.

Example: a = 10

3. **Floating Point Expressions**: One that produces floating point results.

Example: a * b / 2.0

4. **Relational Expressions**: One that returns either true or false value.

Example: c = a>b

5. **Logical Expressions**: One that combines two or more relational expressions and returns a value as True or False.

Example: a>b and y! = 0

6. **Bitwise Expressions**: One that manipulates data at bit level.

Example: x = y&z

7. **Assignment Expressions**: One that assigns a value to a variable.

Example: c = a + b or c = 10

Example

```
a = 20
b = 10
c = 15
d = 5
print ("a:%d b:%d c:%d d:%d" % (a,b,c,d ))
e = (a + b) * c / d
print ("Value of (a + b) * c / d is ", e)
e = ((a + b) * c) / d
print ("Value of ((a + b) * c) / d is ", e)
e = (a + b) * (c / d)
print ("Value of (a + b) * (c / d) is ", e)
e = a + (b * c) / d
print ("Value of a + (b * c) / d is ", e)
```

Output:

a:20 b:10 c:15 d:5

Value of (a + b) * c / d is 90.0

Value of ((a + b) * c) / d is 90.0

Value of (a + b) * (c / d) is 90.0

Value of a + (b * c) / d is 50.0

1.1.14 – Type Conversions

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example - Integers:

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

Example - Floats:

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
```

```
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

Example - Strings:

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

Let's Sum Up

In this section we have studied history of Python Programming Language along with its features. Python is a general purpose high level programming language. Python was developed by Guido Van Rossam in 1989, while working at National Research Institute at Netherlands. The official Date of Birth for Python is Feb 20th 1991. In Python, an identifier (name) must begin with a letter or underscore and can include any number of letters, digits, or underscore. Writing the name of a variable is called declaring a variable whereas assigning a value to a variable is called initialising a variable. In python you can reassign variables as many times as you want to change the value stored in them. The level of indentation groups statements to form a block of statements. A variable of Boolean type can have only one of the two values – True or False.

The input function prompts the user to provide some information on which the program can work and give the result. The print statement is used to display the output screen. Comments are non-executable statements in a program. They are just added to describe the statements in the program code.

Check your progress - QUIZ

1. Which of the following is not a data type?
   a. String b. Numeric c. Array d. Tuples
2. Which character is used for commenting in Python?

a. # b. ! c. @ d. *

3. Which is not a reserved keyword in Python?

a. Insert b. Pass c. Class d. Lambda

4. What is the output of >>> 4+?

a. 4+ b. 4 c. 5 d. Invalid syntax

5. Which of the following is the floor division operator?

a. / b. % c. // d. \\

6. What will be the output of str[0:4] if str="Hello"?

a. 'Hello' b. 'H' c. 'Hel' d. 'Hell'

7. Which of the following is used to find the first index of search string?

a. .find("string") b. .search("string") c. ("string").find d. ("string").search

8. Which of the following is used to access single character of string?

a. [:] b. () c. [.] d. []

9. Which of the following will be printed? x =4.5 y =2 print x//y

a. 2.0 b. 2.25 c. .25 d. 0.5

10. _____ an integer value that represents an element in a sequence

a. index b.item c.list d.id

SECTION 1.2: PYTHON ARRAY

An array is a data structure that stores values of same data type. To use arrays in python language, you need to import the standard array module. This is because array is not a fundamental data type like strings, integer etc. Here is how you can import array module in python:

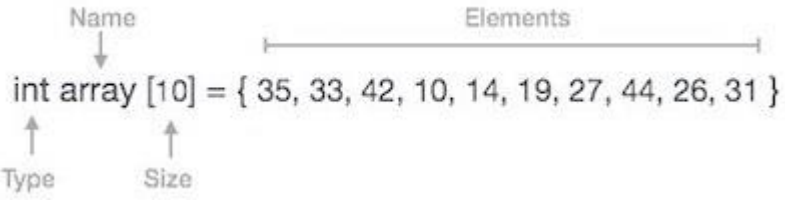
```
from array import *
```

1.2.1 -Defining and processing array

Array is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array.

- **Element**− Each item stored in an array is called an element.

- **Index** − Each location of an element in an array has a numerical index, which is used to identify the element.



- Index starts with 0.
- Array length is 10 which means it can store 10 elements.
- Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.

Python's standard library has array module. The array class in it allows you to construct an array of three basic types, integer, float and Unicode characters.

Syntax

import array

obj = array.array(typecode[, initializer])

- **typecode** − The typecode character used to create the array.
- **initializer** − array initialized from the optional value, which must be a list, a bytes-like object, or iterable over elements of the appropriate type.

Return type

The array() constructor returns an object of array.array class

Type Code	Description
B	Represents signed integer of size 1 byte
B	Represents unsigned integer of size 1 byte
C	Represents character of size 1 byte

u	Represents Unicode character size 2 bytes
h	Represents signed integer of size 2 bytes
H	Represents unsigned integer of size 2 bytes
i	Represents signed integer of size 2 bytes
I	Represents unsigned integer of size 2 bytes
w	Represents unicode character of size 4 bytes
1	Represents signed integer of size 4 bytes
L	Represents unsigned integer of size 4 bytes
f	Represents floating point of size 4 bytes
D	Represents floating point of size 8 bytes

Example

```
from array import *

array1 = array('i', [10,20,30,40,50])

for x in array1:

print(x)
```

OUTPUT:

```
10

20

30

40

50
```

1.2.2 - Array methods

The various methods that can be performed in an array are :

- **Traverse** − Print all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.

Accessing Array Element

Example : element is inserted at index position 1

```
from array import*

array1 =array('i', [10,20,30,40,50])

print (array1[0])

print (array1[2])
```

Output

```
10
30
```

Insertion Operation

Insert operation is to insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array. Here, we add a data element at the middle of the array using the python in-built insert() method.

Example

```
from array import*

array1 =array('i', [10,20,30,40,50])

array1.insert(1,60)
```

```
for x in array1:
    print(x)
```

Output

```
10
60
20
30
40
50
```

Deletion Operation

Deletion refers to removing an existing element from the array and re-organizing all elements of an array. Here, we remove a data element at the middle of the array using the python in-built remove() method.

Example

```
from array import*
array1 =array('i', [10,20,30,40,50])
array1.remove(40)
for x in array1:
    print(x)
```

Output

```
10
20
30
50
```

Search Operation

You can perform a search for an array element based on its value or its index. Here, we search a data element using the python in-built index() method. When we compile and execute the below program, it produces the following result which shows the index of the element. If the value is not present in the array then the program returns an error.

Example

```
from array import*
array1 =array('i',[10,20,30,40,50])
print(array1.index(40))
```

Output

3

Update Operation

Update operation refers to updating an existing element from the array at a given index. Here, we simply reassign a new value to the desired index we want to update.

Example

```
from array import*
array1 =array('i', [10,20,30,40,50])
array1[2] =80
for x in array1:
    print(x)
```

Output

```
10
20
80
```

40

50

Let Us Sum Up

In this section, the learners had the introduction to array concepts, how to create an array and the different types of array because an array is a collection of elements of same type. The way to insert an element into array and how to access the elements in the array were discussed. The various methods that are available for array processing were also mentioned like how to pop an array element.

Check your progress - QUIZ

11. from _____ import * statement used for creating a array
12. The type code used to create floating point number in a array is ____
13. The array index starts from ____
14. To search an element in an array use _____
15. To remove an element in an array ____ is used

UNIT SUMMARY

In this unit we have studied the history and various features of Python Programming Language which made so popular to be used in current upcoming applications by the developer. Python is recommended as first programming language for beginners. Python is an example of Dynamically typed programming language. The literal constants that we can be used directly in Programming Language which made so popular to be used in current upcoming applications by the developer. In Python, the Hash character (#) is used for commenting. Codes or texts that come after the hash character are not considered as a part of the program. We briefly discuss how Operators work using the program code for each operator in Python.

GLOSSARY

- COMMENT: The part of the program not executed by the interpreter. It is used by other persons to understand the program thoroughly.
- DICTIONARY: A mapping of keys to their corresponding values.
- FLOATING POINT: A type of numeral that has a fractional part.
- INDEX: An integer value that represents an element in a sequence.
- INTEGER: A type of numeral that represents whole numbers including negative numbers.
- ITEM: An element or a value in a series.
- ITERATION: The repetition of a set of statements or a piece of code.
- KEYWORD: A word that is reserved in a programming language for a specific purpose. We cannot use keywords such as if and else as variable names.
- OPERAND: The value on which an operator operates

SELF – ASSESSMENT QUESTIONS

1. List out the features of Python Programming
2. Briefly describe the datatypes in python.
3. Describe the different types of operators in Python

EXERCISES

1. What is the output of print list[2] when list = [‘abcd’, 2.23, ‘john’]?
2. How will you convert a string to an integer in Python?
3. What are the uses of //, **, *= operators in Python?
4. Identify the datatype is best suitable to represent the following data values
   a)Number of days in the year
   b)The circumference of a rectangle
   c)Yours father salary
   d)Distance between moon and earth
   e)Name of your best friend

5. Write a Program to find the square root of a number.
6. Write a program that demonstrates the use of relational operators.
7. How a string can be converted to a number?
8. Write a program to swap the values of two variables.
9. Create a floating point array and find the sum of its numbers.
10. Create an array of string and try to do the following:
    a. Search an element
    b. Insert
    c. Delete an element
    d. update

QUIZ - ANSWERS

1. c.array
2. a. #
3. a.insert
4. d.invalid syntax
5. c.//
6. a.'Hello'
7. a.find("string")
8. d.[]
9. a.2.0
10. a.index
11. array
12. f
13. 0
14. index()
15. remove()

SUGGESTED READINGS

1. Yashavant Kanetkar and Aditya Kanetkar, "Let us Python Solutions", -bpb publications, 6th Edition, 2023

2. Ralph T.Burwell," Fundamentals of Python: Basics of Python coding and step-by-step instructions for complete novices" Kindle Edition

3. David Amos, Dan Bader, Joanna Jablonski · ",PythonBasics:A Practical Introduction to Python 3", Real Python (Realpython.Com) Fourth Edition, 2021

OPEN SOURCE E-CONTENT LINKS

- https://www.python.org/about/gettingstarted/
- https://www.tutorialspoint.com/python/python_literals.htm
- https://www.w3schools.com/python/
- https://docs.python.org/3/tutorial/index.html
- https://en.wikipedia.org/wiki/Python_(programming_language)

REFERENCES

1. Reema Thareja,""Python Pogramming using Problem Solving approach", Oxford Higher Education,

2. Kenneth A. Lambert, "Fundamentals of Python- First Programs", CENGAGE Publication.

3. VamsiKurama, "Python Programming: A Modern Approach", Pearson Education.

4. Mark Lutz, "Learning Python", Orielly.

5. E Balagurusamy , "Problem Solving and Python Programming", McGraw Hill Education (India) Private Limited,

PYTHON PROGRAMMING

UNIT 2- PYTHON PROGRAMMING

Control Statements: Selection/Conditional Branching statements: if, if-else, nested if and if-elif-else statements. Iterative Statements: while loop, for loop, else suite in loop and nested loops. Jump Statements: break, continue and pass statements.

Control and Iterative Statements

UNIT OBJECTIVES

In this unit, the learners will have elaborative idea about the declaration of Control Statements, Loop statements and Jump statements. The different types of selection or conditional branching statements like if, if-else, nested if, if-elif-else will be discussed with simple example. The two types of iterative statements like for and while will be discussed with else suite in loop / nested loop which is a new concept in python language. The unit ends its discussion with the use and scope of jump statements like break, continue and pass.

SECTION 2.1: CONTROL STATEMENTS

Python supports a set of control flow statements that you can integrate into your program. The statements inside your Python program are generally executed sequentially from top to bottom in the order that they appear. Apart from sequential control flow statements, you can employ decision making and looping control flow statements to break up the flow of execution thus enabling your program to conditionally execute particular block of code. The term control flow details the direction the program takes.

The control flow statements in Python Programming Language are:

1. **Sequential Control Flow Statements** :This refers to the line by line execution, in which the statements are executed sequentially, in the same order in which they appear in the program.
2. **Decision Control Flow Statements** :Depending on whether a condition is True or False, the decision structure may skip the execution of an entire block

of statements or even execute one block of statements instead of other (if, if…else and if…elif…else).

3. **Loop Control Flow Statements** : This is a control structure that allows the execution of a block of statements multiple times until a loop termination condition is met (for loop and while loop). Loop Control Flow statements are also called Repetition statements of Iteration Statements.

2.1.1– Selection/Conditional Branching statements

The decision control statements usualy jumps, from one part of the code to another depending on whether a particular condition is satisfied or not. That is, they allow you to execute statements selectively based on certain decisions. Such type of decision control statements are known as *selection control statements or conditional branching statements.* Python language supports different types of conditional branching statements which are as follows:

1. If statement
2. If-else statement
3. Nested if statement
4. If-elif-else statement

2.1.1.1 – if statements

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

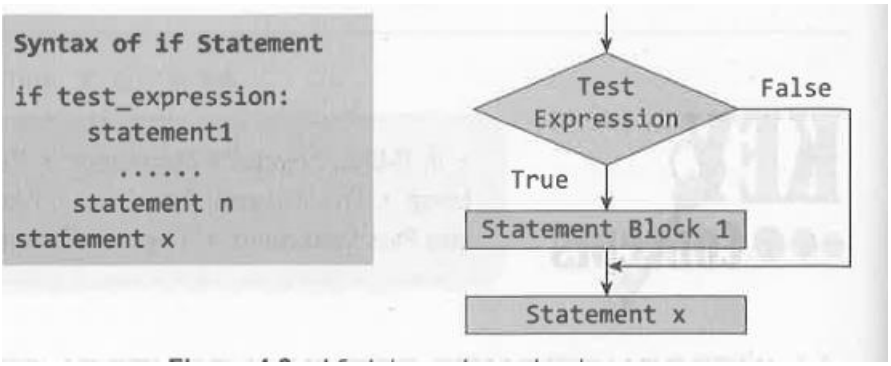


Fig. if statement construct

Example

```
num = int(input("enter the number:"))
if num%2 == 0:
    print("The Given number is an even number")
```

Output:

```
enter the number: 10
The Given number is an even number
```

Example : Program to print the largest of the three numbers.

```
a = int (input("Enter a: "));
b = int (input("Enter b: "));
c = int (input("Enter c: "));
if a>b and a>c:
   print ("From the above three numbers given a is largest");
if b>a and b>c:
   print ("From the above three numbers given b is largest");
if c>a and c>b:
   print ("From the above three numbers given c is largest");
```

Output:

```
Enter a: 100
Enter b: 120
Enter c: 130
From the above three numbers given c is largest
```

2.1.1.2 – if - else statements

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

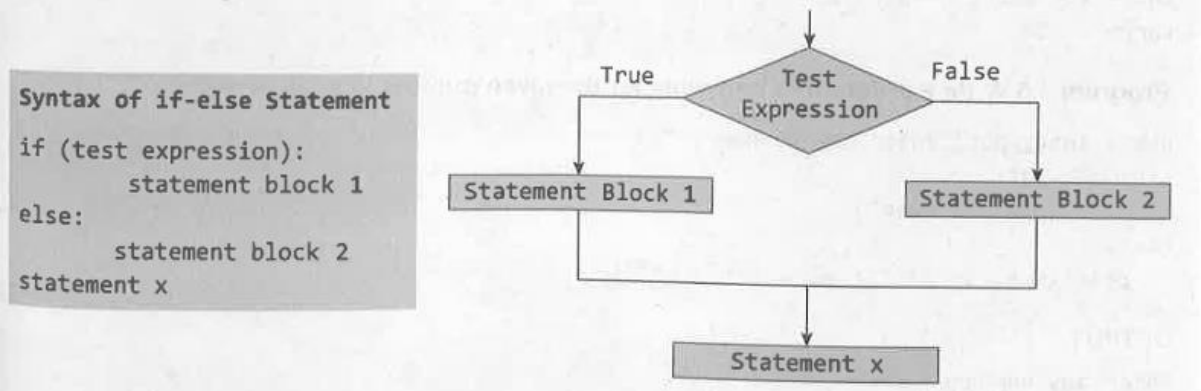


Fig. if-else statement construct

Example : Program to check whether a person is eligible to vote or not.

```
age = int (input("Enter your age: "))
if age>=18:
   print("You are eligible to vote !!");
else:
   print("Sorry! you have to wait !!");
```

Output:

```
Enter your age: 90
You are eligible to vote !!
```

Example : Program to check whether a number is even or not.

```
num = int(input("enter the number:"))
if num%2 == 0:
   print("The Given number is an even number")
else:
   print("The Given Number is an odd number")
```

Output:

```
enter the number: 10
The Given number is even number
```

2.1.1.3 – nested – if statements

Python supports **nested if statements** which means we can use a conditional **if** of else...if statement inside an existing **if statement**. There may be a

situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested **if** construct. In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

```
if expression1:
        statement(s)
if expression2:
        statement(s)
else:
        if expression3:
            statement(s)3
        else
            statement(s)
```

Example

```
num=8
print ("num = ",num)
if num%2==0:
        if num%3==0:
                print ("Divisible by 3 and 2")
        else:
                print ("divisible by 2 not divisible by 3")
else:
        if num%3==0:
                print ("divisible by 3 not divisible by 2")
        else:
                print ("not Divisible by 2 not divisible by 3")
```

Output

```
num = 8
divisible by 2 not divisible by 3
num = 15
```

divisible by 3 not divisible by 2

num = 12

Divisible by 3 and 2

num = 5

not Divisible by 2 not divisible by 3

2.1.1.4 – if-elif-else statements

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.The syntax of the elif statement is given below.

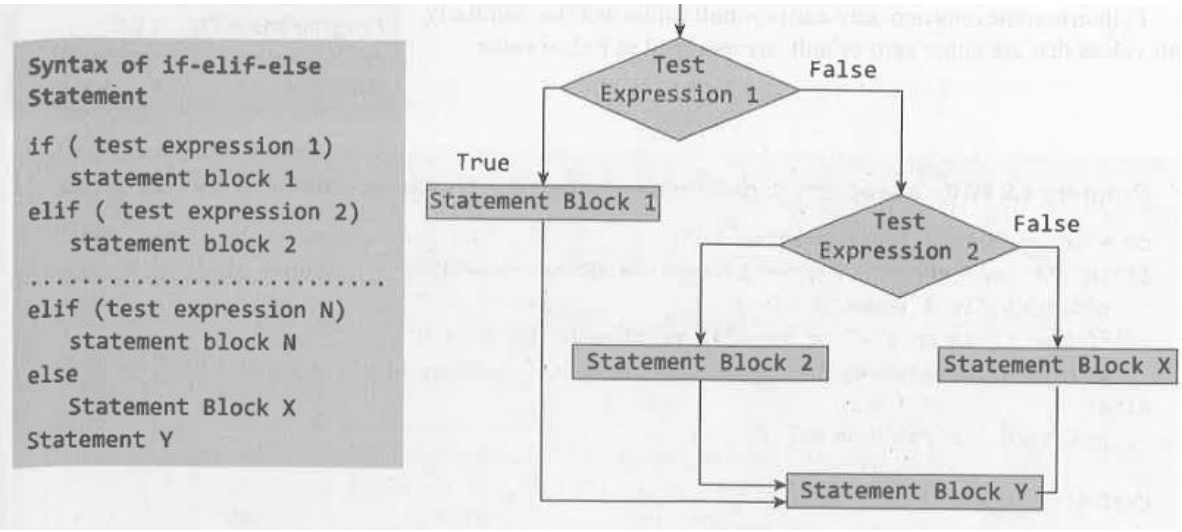


Fig. if-elif-else statement construct

Example 1

```
number = int(input("Enter the number?"))
if number==10:
    print("The given number is equals to 10")
elif number==50:
    print("The given number is equal to 50");
```

```
elif number==100:
    print("The given number is equal to 100");
else:
    print("The given number is not equal to 10, 50 or 100");
```

Output:

```
Enter the number?15
The given number is not equal to 10, 50 or 100
```

Example

```
marks = int(input("Enter the marks? "))
if marks > 85 and marks <= 100:
        print("Congrats ! you scored grade A ...")
elif marks > 60 and marks <= 85:
        print("You scored grade B + ...")
elif marks > 40 and marks <= 60:
        print("You scored grade B ...")
elif (marks > 30 and marks <= 40):
        print("You scored grade C ...")
else:
        print("Sorry you are fail ?")
```

Output:

```
Enter the marks? 89
Congrats !you scored grade A
```

2.1.2 – Iterative statements

Python supports basic loop strcures through iterative statements. *Iterative statements* are decision control statements that are used to repeat the execution of a list of statements. Python language supports two types of iterative statements- while loop and for loop.

2.1.2.1 while loop

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given boolean expression is true.

Syntax

```
while expression:
    statement(s)
```

Here, statement(s) may be a single statement or a block of statements with uniform indent. The condition may be any expression, and true is any non-zero value. The loop iterates while the boolean expression is true.

As soon as the expression becomes false, the program control passes to the line immediately following the loop. The following flow diagram illustrates the **while** loop

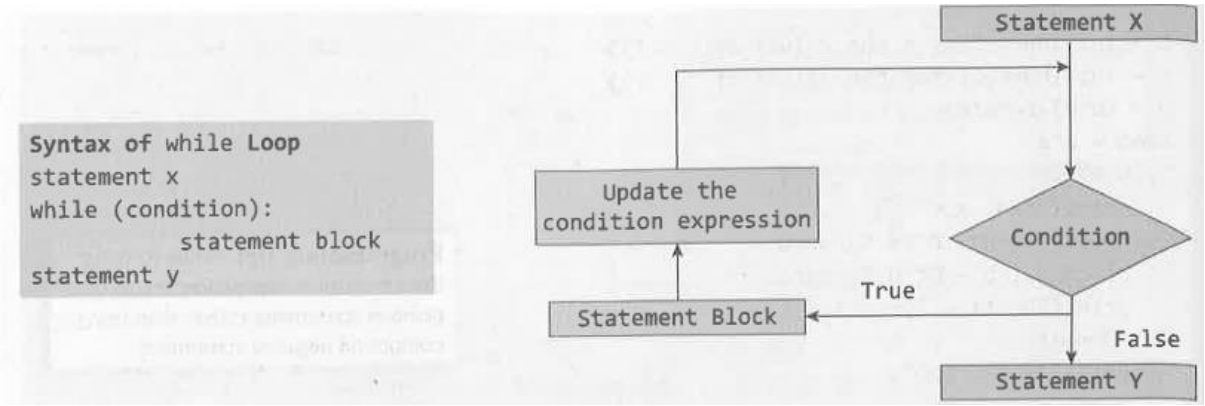


Fig The while loop construct

Example

```
number = 1
while number <= 3:
    print(number)
    number = number + 1
```

Output

```
1
2
3
```

Example

```
# Calculate the sum of numbers until user enters 0
number = int(input('Enter a number: '))
total = 0
# iterate until the user enters 0
while number != 0:
    total += number
    number = int(input('Enter a number: '))
print('The sum is', total)
```

Output

```
Enter a number: 3
Enter a number: 2
Enter a number: 1
Enter a number: -4
Enter a number: 0
The sum is 2
```

2.1.2.2 for loop

In Python, the *for* Statement runs the code block each time it traverses a series of elements. The loop_*control_var* is the parameter that determines the element's value within the iterable sequence on each iteration. When a sequence contains expression statements, they are processed first. The first element in the sequence is then assigned to the iterating variable *iterating_variable*. From that point onward, the planned block is run. Each element in the sequence is assigned to iterating_variable during the statement block until the sequence as a whole is completed. Using indentation, the contents of the Loop are distinguished from the remainder of the program.

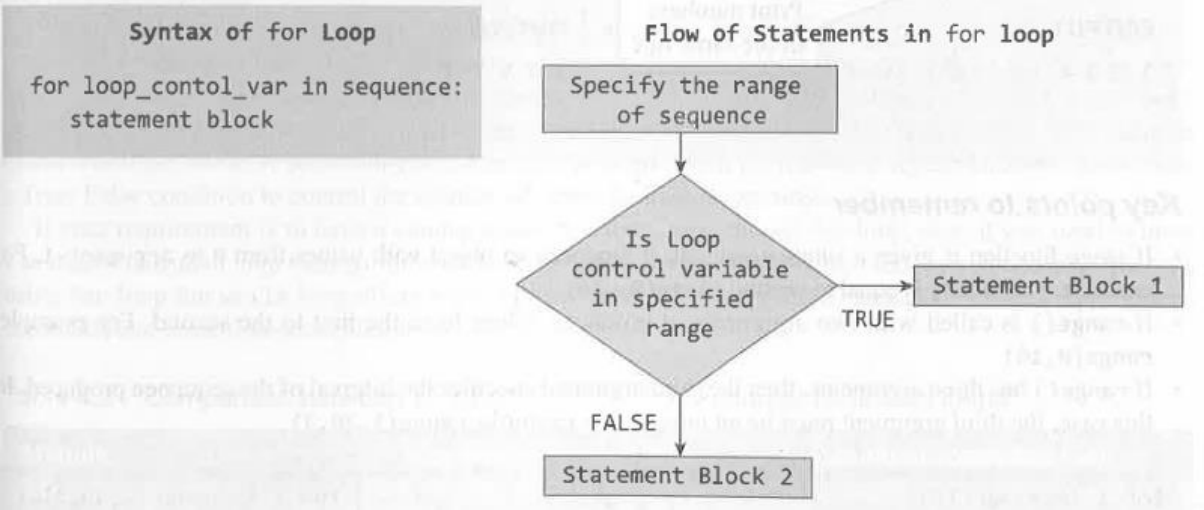


Fig The for loop construct

Example -find the sum of squares of each element of the list using for loop

```
# creating the list of numbers
numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
# initializing a variable that will store the sum
sum = 0
# using for loop to iterate over the list
for num in numbers:
    sum = sum + num ** 2
print("The sum of squares is: ", sum)
```

Output:

```
The sum of squares is:  774
```

Example

```
languages = ['Swift', 'Python', 'Go']
# access elements of the list one by one
for i in languages:
    print(i)
```

Output:

```
Swift
Python
Go
```

Example

```
language = 'Python'
for x in language: # iterate over each character in language
    print(x)
```

Output

```
P
y
t
h
o
n
```

2.1.2.3 else suite in loop and nested loops

Python supports having an **else** statement associated with a **while** loop statement. If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false before the control shifts to the main line of execution.

Example –

for letter in “HELLO”:
  print(letter, end=” “)
else:
  print(“\nDone”)
Output:
H E L L O
Done
	i=1
	while(i<0):
	  print(i)
	  i = i-1
	else:
	  print(i,” is not negative so loop did not execute”)
	Output
	1 is not negative so loop did not execute

Let's Sum Up

In this section, learners studied about the control statements in Python Programming. The control statements include conditional and iterative statements. There are four conditional statements like if, if-else, nested if and if-elif-else statements. All the four statements are explained with simple example and their expected output which helps the learner to understand the concept easily. Under iterative statements while loop and for loop concepts are discussed.

Check your progress – QUIZ 1

1. _____ is a control structure that allows the execution of a block of statements multiple times until a condition is met.

   a. Sequential Control Flow Statements

   b. Decision Control Flow Statements

   c. Loop Control Flow Statements

   d. Jump Control Flow Statements

2. An if statement can also be followed by an _____statement which is optional.

   a. Else b. then c. else d.Then

3. _____ is a short form of "else if statement.

   a.elif b.Elif c.Elseif d.ifelse

4. _____ can be used when the number of times the statements in loop has to be executed is not know in advance

   a. For b.for c.while d.While

5. for i in range(0,20):

   print(____)

fill the blank to print only odd values in the range

SECTION 2.2: JUMP STATEMENTS

2.2.1 – break statement

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop. The break is commonly used in the cases where we need to break the loop for a given condition.

Example

```
my_list = [1, 2, 3, 4]
count = 1
for item in my_list:
    if item == 4:
    print("Item matched")
    count += 1
    break
print("Found at location", count)
```

Output:

```
Item matched
Found at location 2
```

2.2.2 – continue statement

Python continue keyword is used to skip the remaining statements of the current loop and go to the next iteration. In Python, loops repeat processes on their

own in an efficient way. However, there might be occasions when we wish to leave the current loop entirely, skip iteration, or dismiss the condition controlling the loop.

We use Loop control statements in such cases. The continue keyword is a loop control statement that allows us to change the loop's control.

Example

```
for iterator in range(10, 21):
 # If iterator is equals to 15, loop will continue to the next iteration
        if iterator == 15:
                continue
           # otherwise printing the value of iterator
print( iterator )
```

Output:

10
11
12
13
14
16
17
18
19
20

2.2.3 – pass statement

The pass statement serves as a placeholder for future code, preventing errors from empty code blocks.It's typically used where code is planned but has yet to be written.

Example

```
def future_function():
```

```
        pass
# this will execute without any action or error
future_function()
```

Let's Sum Up

In this section, learners studied about the control statements which includes conditional and iterative statements. The conditional statements includes if, if-else, nested if and if-elif-else statements. All the four statements are explained with simple example along with their expected output. The while..loop and for..loop comes under iterative statements concepts. In second part of the unit, the Jump statements like pass, continue and break in Programming Language were discussed. The jump statements are used to alter the program flow. Using break or continue outside a loop causes an error. Pass statement is used when a statement is required syntactically but no command or code has to be executed.When the compiler encounters a break statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the outside the loop.

Check your progress QUIZ-2

6. To specify an empty body of a for loop, you will use ____ statement.
   a. Break b.break c.continue d.**pass**
7. Whenever the _____statement is encountered, the execution control immediately jumps to the first instruction following the loop.
   a.Break b.**break** c.continue d.pass
8. To pass control to the next iteration without exiting the loop, use the _____statement.
   a.Break b.break c.**continue** d.pass
9. Which statement is used to terminate the execution of the nearest enclosing loop in which it appears?
   a.Break b.**break** c.continue d.pass
10. Which statement indicates a NOP?
    a.Break b.break c.continue d.**pass**

UNIT SUMMARY

In this section, the learners learned the Jump statements like pass, continue and break in Programming Language. The jump statements are used to alter the program flow. Using break or continue outside a loop causes an error. When the compiler encounters a continue statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop. Pass statement is used when a statement is required syntactically but no command or code has to be executed.

GLOSSARY

Conditional branching statements Statements that helps to jump from one part of the program to another depending on whether a particular conditions is satisfied or not.

If-else-if statement Decision control statement that works in the same way as a normal if statement. It also known as nested if construct.

If-else statement Decision control statement in which first the test expression is evaluated. If the expression is True, if block is executed and else block is skipped. Otherwise, if the expression is false, else block is executed and if the block is ignored.

If-statement Simplest form of decision control statement that is frequently used in decision making.

SELF – ASSESSMENT QUESTIONS

1. Write a program to find whether a number is even or odd
2. Write a program to check the largest among the given three numbers
3. Write a Python program to check if the input year is a leap year or not
4. Write a program to print the prime numbers for given range

5. Write a program to demonstrate while loop with else.

EXERCISES

1. Write a short note on conditional branching statements supported by Python.
2. Explain the syntax of for loop.
3. Explain the utility of break statement with the help an example.
4. Explain the utility of continue statement with the help an example.
5. Differentiate between pass and continue statement.

QUIZ - ANSWERS

QUIZ - 1 & 2

1. c.Loop Control Flow Statements
2. c.else
3. a.elif
4. c.while
5. 2*i+1
6. d.pass
7. b.break
8. c.continue
9. b.break
10. d.pass

SUGGESTED READINGS

1. Yashavant Kanetkar and Aditya Kanetkar, “Let us Python Solutions”, -bpb publications, 6th Edition, 2023

2. Ralph T.Burwell," Fundamentals of Python: Basics of Python coding and step-by-step instructions for complete novices" Kindle Edition

3. David Amos, Dan Bader, Joanna Jablonski · ",PythonBasics:A Practical Introduction to Python 3", Real Python (Realpython.Com) Fourth Edition, 2021

OPEN SOURCE E-CONTENT LINKS

- https://www.python.org/about/gettingstarted/
- https://www.w3schools.com/python/
- https://docs.python.org/3/tutorial/index.html
- https://www.geeksforgeeks.org/python-programming-language/

REFERENCES

1. Reema Thareja,""Python Programming using Problem Solving approach", Oxford Higher Education.
2. Dr. R. NageswaraRao, "Core Python Programming", First Edition, 2017, Dream tech Publishers.
3. VamsiKurama, "Python Programming: A Modern Approach", Pearson Education.
4. Mark Lutz, "Learning Python", Orielly.
5. E Balagurusamy , "Problem Solving and Python Programming", McGraw Hill Education (India) Private Limited.

PYTHON PROGRAMMING

UNIT 3- PYTHON PROGRAMMING

Functions: Function Definition – Function Call – Variable Scope and its Lifetime-Return Statement. Function Arguments: Required Arguments, Keyword Arguments, Default Arguments and Variable Length Arguments- Recursion. Python Strings: String operations- Immutable Strings - Built-in String Methods and Functions - String Comparison. Modules: import statement- The Python module – dir() function – Modules and Namespace – Defining our own modules

Functions, String, and Modules

UNIT OBJECTIVES

In this unit, the learners will understand the meaning and scope of functions, Strings and modules in python programming. It will discuss the way to define a function / call a function. The various types of arguments supported by function concept will be discussed. The variable scope and life time which is very important to keep track is discussed. It also describes the role of return statement in a function. In Python String place a major role. The way its declared, accessed, formatted, indexed are also discussed. Finally the role of Module in python will be briefed.

SECTION 3.1: FUNCTIONS

A function is a block of organized and reusable program code that performs a single, specific, and well-defined task. Python enables its programmers to break up a program into functions, each of which can be written more or less independently of the others. Therefore, the code of one function is completely insulated from the codes of the other functions.

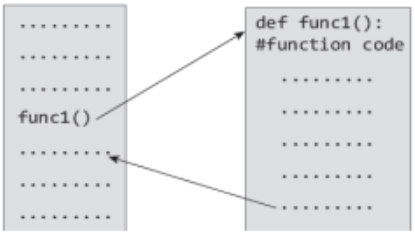


Figure 1: Calling a function

In figure 1 which explains how a function func1() is called to perform a well defined task. As soon as func1() is called, the program control is passed to the first statement in the function. All the statements in the function are executed and then the program control is passed to the statement following the one that called the function.

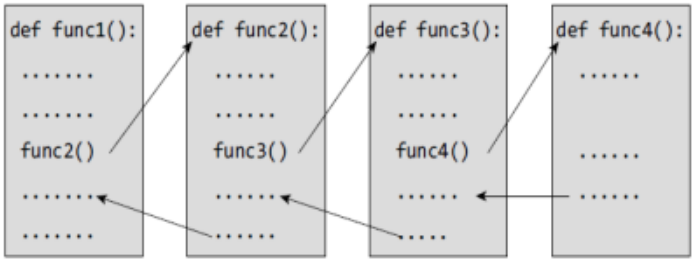


Figure 2: Function calling another function

In figure 2 func1() calls function named func2(). Therefore, func1() is known as the calling function and func2() is known as the called function. The moment the compiler encounters a function call, instead of executing the next statement in the calling function, the control jumps to the statements that are a part of the called function. After called function is executed, the control is returned back to the calling program. It is not necessary that the func1() can call only one function, it can call as many functions as it wants and as many times as it wants. For example, a function call placed within for loop or while loop may call the same function multiple times until the condition holds true.

Hence, it becomes important to organize the program into more manageable units. Further, it makes the code reusable. There are three types of functions in python-

1. **Built-in functions** – these are the functions which are already defined in the language. Example print( ), max( ), input( ), etc.

2. **User Defined functions**- these are the functions that can be created or defined by the users according to their needs.

3. Anonymous functions

3.1.1– Function Definition

A function can be defined using def statement and giving suitable name to a function by following the rules of identifiers. The process of defining a function is called function definition.

Syntax

def function_name(list_of_parameters) :
statement (s)

- **function_name()** - is the name of the function. A function name must be followed by a set of parenthesis. Any name can be given to a function following the rules of identifiers.
- **List_of _parameters** –is an optional field which is used to pass values or inputs to a function. It can be none or a comma separated list of variables.
- **Statements(s)** – is a set of statements or commands within the function. Each time the function is called, all the statements will be executed. These statements together form the body of a function.

3.1.2 Function Call

Even if the function is defined, it can never be executed till the time it is called. Hence, for using a function, it must be called using function call statement. A function can be called by its name with set of parenthesis and optional list of arguments.

Syntax

function_name(list_of_arguments)

Example

```
def first_function():          #function definition
    print("HELLO WORLD …!")
first_function()               #function calling
```

OUTPUT:

HELLO WORLD …!

Function needs to be defined only once, but it can be called any number of times by using calling statements. They can not only be called in the same program, but they can also be called in different programs. Another example shown below is the program having a function to calculate factorial of a number.

Example

```
def factorial():
    fact=1
    n=int(input("enter a number :")
    for i in range(1, n+1):
        fact= fact + i
    print("factorial of ", n ",is ",fact)
factorial()
```

Output:

Enter a number: 4

Factorial of 4 is 24

3.1.3–Variable Scope and its lifetime

Part of a code in which a variable can be accessed is called Scope of a variable. Scope of a variable can be global or local.

- **Local variables** are the variable created within a function's body or function's scope. They cannot be accessed outside the function. Their scope is only limited to the function in which they are created.

- **Global variables** are the variable created outside any function. Their scope is global, hence can be used anywhere. Global variables can also be created within function using keyword global.

Example: Use of local and global variables.

```
num1 = 10    # global variable
print("Global variable num1 = ", num1)
def func(num2):           # num2 is function parameter
        print("In Function - Local Variable num2 = ",num2)
        num3 = 30         #num3 is a local variable
        print("In Function - Local Variable num3 = ",num3)
func(20)       #20 is passed as an argument to the function
print("num1 again = ", num1)        #global variable is being accessed
#Error- local variable can't be used outside the function in which it is defined
print("num3 outside function = ", num3)
```

OUTPUT

```
Global variable num1 = 10
In Function - Local Variable num2 = 20
In Function - Local Variable num3 = 30
num1 again =  10
num3 outside function =
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 12, in <module>
    print("num3 outside function = ", num3)
NameError: name 'num3' is not defined
```

Programming Tip: Variables can only be used after the point of their declaration

Comparison between global and local variables

Global variables	Local variables
They are defined in the main body of the program file.	They are defined within a function and is local to that function.
They can be accessed throughout the program life.	They can be accessed from the point of its definition until the end of the block in which it is defined.
Global variables are accessible to all functions in the program.	They are not related in any way to other variables with the same names used outside the function

3.1.4 Return Statement

A function may or may not have a return[expression] statement. That is, the return statement is optional. You can assign the function name to a variable After writing the code statements, the block is ended with a return statement whose syntax

is return [expression].If you want to return more than one value, separate the values using commas. The default return value is NONE.

Syntax

return [expression]

The expression is written in brackets because it is optional. If the expression is present, it is evaluated and the resultant value is returned to the calling function. However, if no expression is specified then the function will return none. The return statement is used for two things.

- Return a value to the caller
- To end and exit a function and go back to its caller

Example

```
def factorial(N):
        fact=1
        for i in range(1, n+1):
                fact= fact + i
return(fact)
n=int(input("enter a number :")
fact=factorial(n)
print("factorial of ", n ",is ",fact)
```

OUTPUT

Enter a number :4

Factorial of 4 is 24

3.1.5 Function Arguments

Parameters and arguments are the values or expressions passed to the functions between parentheses. As we have seen in earlier sections, many of the built in functions need arguments to be passed with them: the math.cos() function takes a number, i.e., the value of the angle as an argument. Many functions require

two or more arguments to be passed such as the power function in math module math.pow(), where we have to pass two arguments, the base and the exponent.

The value of the argument is always assigned to a variable known as parameter. At the time of function definition, we have to define some parameters if that function requires some arguments to be passed at the time of calling.

There can be four types of formal arguments using which a function can be called which are as follows:

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

3.1.5.1 Required Arguments

Required arguments are those supplied to a function during its call in a predetermined positional sequence. The number of arguments required in the method call must be the same as those provided in the function's definition.

We should send two contentions to the capability() all put together; it will return a language structure blunder, as seen beneath.

Example

```
# Defining a function
def function( n1, n2 ):
    print(“number 1 is: ”, n1)
    print(“number 2 is: ”, n2)

# Calling function and passing two arguments out of order
#we need num1 to be 20 and num2 to be 30
print( ”Passing out of order arguments” )
function( 30, 20 )
```

```
# Calling function and passing only one argument
print( ”Passing only one argument” )
try:
   function( 30 )
except:
   print( ”Function needs two positional arguments” )
```

Output:

```
Passing out of order arguments
number 1 is:  30
number 2 is:  20
Passing only one argument
Function needs two positional arguments
```

3.1.5.2 Keyword Statement

Keyword arguments are linked to the arguments of a called function. While summoning a capability with watchword contentions, the client might tell whose boundary esteem it is by looking at the boundary name.

We can eliminate or orchestrate specific contentions in an alternate request since the Python translator will interface the furnished watchwords to connect the qualities with its boundaries. One more method for utilizing watchwords to summon the capability() strategy is as per the following:

Example

```
 # Defining a function
def function( n1, n2 ):
   print(“number 1 is: ”, n1)
   print(“number 2 is: ”, n2)

# Calling function and passing arguments without using keyword
print( ”Without using keyword” )
function( 50, 30)
```

```
# Calling function and passing arguments using keyword
print( ”With using keyword” )
function( n2 = 50, n1 = 30)
```

Output:

```
Without using keyword
number 1 is:  50
number 2 is:  30
With using keyword
number 1 is:  30
number 2 is:  50
```

3.1.5.3 Default Statement

A default contention is a boundary that takes as information default esteem, assuming that no worth is provided for the contention when the capability is called. The following example demonstrates default arguments.

Example

```
# defining a function
def function( n1, n2 = 20 ):
print(“number 1 is: “, n1)
print(“number 2 is: “, n2)
# Calling the function and passing only one argument
print( “Passing only one argument” )
function(30)
# Now giving two arguments to the function
print( “Passing two arguments” )
function(50,30)
```

Output:

```
Passing only one argument
number 1 is:  30
```

```
number 2 is:  20
Passing two arguments
number 1 is:  50
number 2 is:  30
```

3.1.5.4 Variable length Arguments

We can involve unique characters in Python capabilities to pass many contentions. However, we need a capability. This can be accomplished with one of two types of characters:

“args” and “kwargs” refer to arguments not based on keywords.

Example

```
# Defining a function
def function( *args_list ):
    ans = []
    for l in args_list:
        ans.append( l.upper() )
    return ans
# Passing args arguments
object = function('Python', 'Functions', 'tutorial')
print( object )

# defining a function
def function( **kargs_list ):
    ans = []
    for key, value in kargs_list.items():
        ans.append([key, value])
    return ans
# Paasing kwargs arguments
object = function(First = ”Python”, Second = ”Functions”, Third = ”Tutorial”)
print(object)
```

Output:

['PYTHON', 'FUNCTIONS', 'TUTORIAL']

[['First', 'Python'], ['Second', 'Functions'], ['Third', 'Tutorial']]

3.1.6 Recursion

Recursion is the process of defining something in terms of itself.If a function, procedure or method calls itself, it is called recursive. In Python, we know that a function can call another function, but it is also possible that a function calls itself. Let us look at an example of a recursive function by computing the factorial of a number. The factorial of any number is defined by multiplying all the integers from 1 to that number. For example, the factorial of 5 is 1*2*3*4*5 = 120.

Example

```
def factorial(x):
    """This is a recursive function to find the factorial of an integer"""
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))
num = 10
print("The factorial of", num, "is", factorial(num))
```

OUTPUT:

The factorial of 10 is 3628800

Let's Sum Up

In this section we have seen the role of a function definition in Python Programming. Functions are self-contained programs that perform some particular tasks. Once a function is created by the programmer for a specific task, this function can be called anytime to perform that task. Suppose, we want to perform a task several times, in such a scenario, rather than writing code for that particular task repeatedly, we create a function for that task and call it when we want to perform the task. Each function is given a name, using which we call it. A function may or may not return a value. There are many built-in functions provided by Python such as

dir(), len(), abs(), etc. Users can also build their own functions, which are called user-defined functions. If a function, procedure or method calls itself, it is called recursive

Check your progress - QUIZ

1. What are the advantages of using functions?
   a. Reduce duplication of code b. Clarity of code
   c. Reuse code d. All

2. Which keyword is used to define the block of statement in the function?
   a. Function b. def c. func d. pi

3. Which file contains the predefined Python codes?
   a. Function b. Pi c. module d. lambda

4. A function is called using the name with which it was defined earlier, followed by: a. { } b. () c. <> d. [ ]

5. What is the use of the return statement?
   a. exit a function b. null value c. initiate a function d. none

SECTION 3.2: PYTHON STRING

In Python, a **string** is an immutable sequence of Unicode characters. Each character has a unique numeric value as per the UNICODE standard. But, the sequence as a whole, doesn't have any numeric value even if all the characters are digits. To differentiate the string from numbers and other identifiers, the sequence of characters is included within single, double or triple quotes in its literal representation. Hence, 1234 is a number (integer) but '1234' is a string.

Example

```
#Using single quotes
str1 = 'Hello Python'
print(str1)
```

```
#Using double quotes
str2 = ”Hello Python”
print(str2)
#Using triple quotes
str3 = """Triple quotes are generally used for
    represent the multiline or
    docstring"'
print(str3)
```

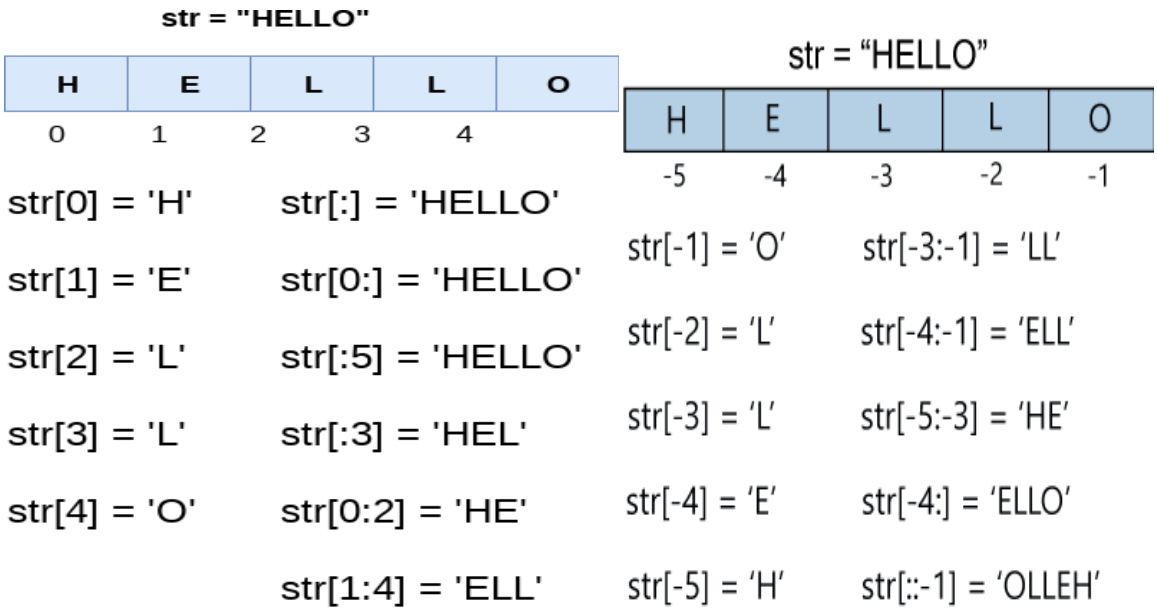
Output:

```
Hello Python
Hello Python
Triple quotes are generally used for
represent the multiline or
docstring.
```

3.2.1 String Operations

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give - HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1
r/R	Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter “r,” which precedes the quotation marks. The “r” can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote	print r'\n' prints \n and print R'\n'prints \n

	mark.	



3.2.2 Immutable Strings

Python strings are immutable which means that once created they cannot be changed. Whenever you try to modify an existing string variable, a new string is created.

Every object in Python is stored in memory. You can find out whether two variables are referring to the same object or not by using the id(). The id() returns the memory address of that object. As both str1 and str2 points to same memory location, they both point to the same object.

Example

```
str1="hello"
print("str1 is :",str1)
print("id of str1 is :", id(str1))
str2="world"
print("str2 is :",str2)
print("id of str1 is :", id(str2))
str1 +=str2
print("str1 after concatenation is :",str1)
```

```
    print("id of str1 is :", id(str1))
    str3=str1
    print("str3 is :",str3)
print("id of str3 is :", id(str3))
```

OUTPUT:

```
str1 is : hello
id of str1 is : 45093344
str2 is : world
id of str2 is : 45093321
str1 after concatenation is : helloworld
id of str1 is : 43861792
str3 is : helloworld
id of str3 is : 43861792
```

3.2.3 Built-in String Methods and Functions

Python includes the following built-in methods to manipulate strings −

Methods with Description
capitalize() :- Capitalizes first letter of string.
casefold():-Converts all uppercase letters in string to lowercase. Similar to lower(), but works on UNICODE characters alos.
center(width, fillchar) :- Returns a space-padded string with the original string centered to a total of width columns.
count(str, beg= 0,end=len(string)) :-Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
decode(encoding='UTF-8',errors='strict') :-Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
encode(encoding='UTF-8',errors='strict') :- Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.
endswith(suffix, beg=0, end=len(string)) :-Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.

expandtabs(tabsize=8) :-Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
find(str, beg=0 end=len(string)) :- Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
format(*args, **kwargs) :-This method is used to format the current string value.
format_map(mapping) :-This method is also use to format the current string the only difference is it uses a mapping object.
index(str, beg=0, end=len(string)) :-Same as find(), but raises an exception if str not found.
isalnum():- Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
isalpha():-eturns true if string has at least 1 character and all characters are alphabetic and false otherwise.
isascii():-Returns True is all the characters in the string are from the ASCII character set.
isdecimal():-Returns true if a unicode string contains only decimal characters and false otherwise.
isdigit():-Returns true if string contains only digits and false otherwise.
isidentifier():-Checks whether the string is a valid Python identifier.
islower():-Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
isnumeric():-Returns true if a unicode string contains only numeric characters and false otherwise.
isprintable():-Checks whether all the characters in the string are printable.
isspace():-Returns true if string contains only whitespace characters and false otherwise.
istitle():-Returns true if string is properly “titlecased” and false otherwise.
isupper():-Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
join(seq) :-Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.

ljust(width[, fillchar]) :-Returns a space-padded string with the original string left-justified to a total of width columns.
lower():-Converts all uppercase letters in string to lowercase.
lstrip():- Removes all leading white space in string.
maketrans():-Returns a translation table to be used in translate function.
partition():-Splits the string in three string tuple at the first occurrence of separator.
removeprefix():-Returns a string after removing the prefix string.
removesuffix():-Returns a string after removing the suffix string.
replace(old, new [, max]) :-Replaces all occurrences of old in string with new or at most max occurrences if max given.
rfind(str, beg=0,end=len(string)) :-Same as find(), but search backwards in string.
rindex(str, beg=0, end=len(string)) :-Same as index(), but search backwards in string.
rjust(width,[, fillchar]) :- Returns a space-padded string with the original string right-justified to a total of width columns.
rpartition():-Splits the string in three string tuple at the ladt occurrence of separator.
rsplit():-Splits the string from the end and returns a list of substrings.
rstrip():-Removes all trailing whitespace of string.
split(str=““, num=string.count(str)) :-Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
splitlines(num=string.count('\n')) :-Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
startswith(str, beg=0,end=len(string)) :-Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.
strip([chars]) :-Performs both lstrip() and rstrip() on string.
swapcase():-Inverts case for all letters in string.

title():-Returns “titlecased” version of string, that is, all words begin with uppercase and the rest are lowercase.
translate(table, deletechars=““) :-Translates string according to translation table str(256 chars), removing those in the del string.
upper():-Converts lowercase letters in string to uppercase.
zfill (width) :-Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).

Following are the **built-in functions** we can use with strings −

Function with Description
len(list) :-Returns the length of the string.
max(list) :-Returns the max alphabetical character from the string str.
min(list) :-Returns the min alphabetical character from the string str.

3.2.4 String Comparison

We use the == operator to compare two strings. If two strings are equal, the operator returns True. Otherwise, it returns False. For example,

Example

```
str1 = “Hello, world!”
str2 = “I love Swift.”
str3 = “Hello, world!”
# compare str1 and str2
print(str1 == str2)
# compare str1 and str3
print(str1 == str3)
```

Output

```
False

True
```

In the above example,

- str1 and str2 are not equal. Hence, the result is False.
- str1 and str3 are equal. Hence, the result is True.

Let's Sum Up

Besides numbers, strings are another important data type. Single quotes or double quotes are used to represent strings. A string in Python can be a series or a sequence of alphabets, numerals and special characters. Similar to C, the first character of a string has an index 0. There are many operations that can be performed on a string. There are several operators such as slice operator ([]) and [:]), concatenation operator (+), repetition operator (*), etc. Slicing is used to take out a subset of the string, concatenation is used to combine two or more than two strings and repetition is used to repeat the same string several times.

Check your progress - QUIZ

6. _____ is a group of characters.
   a.string b.word c.sentence d.line
7. You cannot multiply a string with a floating point number. (True / False)
8. You can print a string without the print function. (True / False)
9. Single quotes and double quotes ina string cannot used within triple quotes. (True / False)
10. Char is a valid data type in python.(True/ False)

SECTION 3.3: MODULES

We have seen that functions help us to reuse a particular piece of code. Module goes a step ahead. It allows you to reuse one or more functions in your programs, even in the programs in which those functions have not been defined. Putting simply, module is a file with a.py extension that has definitions of all.

Functions and variables that you would like to use even in other programs. The program in which you want to use functions or variables defined in the module will simply import that particular module (or .py file). Modules are pre-written pieces of code that are used to perform common tasks like generating random numbers, performing mathematical operations, etc.

The basic way to use a module is to add import module_name as the first line of your program and then writing module_name.var to access functions and values with the name var in the module.

3.3.1– import statement

A module may contain definition for many variables and functions. When you import a module, you can use any variable or function defined in that module. But if you want to use only selected variables or functions, then you can use the *from...import* statement.

For example, in the aforementioned program you are using only the path variable in the sys module, so you could have better written from sys import path.

Example

```
from math import pi
print("PI =", pi)
```

OUTPUT:

```
PI =3.141592653589793
```

To import more than one item from a module, use a comma separated list. For example, to import the value of pi and sqrt() from the math module you can write,

```
from math import pi,sqrt
```

3.3.2 – The Python Module

- We have seen that a Python module is a file that contains some definitions and statements. When a Python file is executed directly, it is considered the main module of a program.

- Main modules are given the special name __main__ and provide the basis for a complete Python program.
- The main module may import any number of other modules which may in turn import other modules. But the main module of a Python program cannot be imported into other module

3.3.3– dir() function

dir() is a built-in function that lists the identifiers defined in a module. These identifiers may include functions, classes and variables. If no name is specified, the dir() will return the list of names defined in the current module.

Example

```
def print_var(x):
        print(x)
        x=10
print_var(x)
print(dir())
```

OUTPUT:

```
10
['__builtins__','__doc__','__name__','__package__','print_var','x']
```

3.3.4 modules and namespace

A namespace is a container that provides a named context for identifiers. Two identifiers with the same name in the same scope will lead to a name clash. In simple terms, Python does not allow programmers to have two different identifiers with the same name. However, in some situations we need to have same name identifiers. To cater to such situations, namespaces is the keyword. Namespaces enable programs to avoid potential name clashes by associating each identifier with the namespace from which it originates.

Example:

```
#module1
def repeat_m(x):
```

```
        return x*3;
#module2
def repeat_m(x):
        return x*3;
import module1
import module2
result=repeat_m(10)  #ambiguous reference for identifier repeat_m
```

Advantages of Modules:

- Python modules provide all the benefits of modular software design. These modules provide services and functionality that can be reused in other programs.
- Even the standard library of Python contains a set of modules. It allows you to logically organize the code so that it becomes easier to understand and use.

3.3.5 – Defining your own module

Every Python program is a module, that is, every file that you save as .py extension is a module. Modules should be placed in the same directory as that of the program in which it is imported. It can also be stored in one of the directories listed in sys.path.

First write these lines in a file and save the file as **mymodule.py**

```
def display():
        print("Hello")
        print("Name of called module is ….", __name__)
str="Welcome to the World of Python !!!"
```

Then open another file (**main.py**) and write the lines of code given below.

```
import mymodule
print("My module str = ", mymodule.str)
mymodule.display()
```

```
print("Name of calling module is ….", __name__)
```

OUTPUT:

```
My module str = Welcome to the World of Python !!!
Hello
Name of called module is …. mymodule
Name of calling module is ….main
```

When a Python file is executed directly, it is considered the main module of a program. Main modules are given the special name __main__ and provide the basis for a complete Python program. The main module may import any number of other modules which may in turn import other modules. But the main module of a Python program cannot be imported into other modules.

Let Us Sum Up

In this section we have studied the literal constants that we can be used directly in Programming Language which made so popular to be used in current upcoming applications by the developer. Python is recommended as first programming language for beginners. Python is an Example of Dynamically typed programming language.

Check your progress - QUIZ

11. Modules are files saved with ____ extension
    a.mod b..py c..pyi d..mod
12. To import sqrt and cos function from the math module write ____
13. It is mandatory to place all import statements at the beginning of a module (True / False)
14. If a particular module is imported more that once, the interpreter will load the module only once. (True / False)
15. A function can be called anywhere within a program (True / False)

UNIT SUMMARY

In this section, the learners learned about the functions, which are self-contained programs that perform some particular tasks. There are many built-in functions provided by Python such as dir(), len(), abs(), etc., and users can also make their own functions which are known as user-defined functions. The block of the function starts with a keyword def after which we write our function name followed by parentheses. A module is a file that contains some predefined Python codes. A module can define functions, classes and variables. It is a collection of related functions grouped together.

Parameters and arguments are the values or expressions that are passed to the functions between the parentheses. In keyword arguments, the caller recognises the arguments by the parameter's names. This type of argument can also be skipped or can also be out of order. In default arguments, we can assign a value to a parameter at the time of function definition that will be considered the default value to that parameter.

GLOSSARY

FUNCTION: Functions are self-contained programs that perform some particular tasks.

FUNCTION OBJECT: A value created by the definition of a function. A variable which is the name of the function refers to the function object.

HEADER: The very first line of the function definition.

BODY: The block of statements inside the function definition.

PARAMETER: The variables used to pass some values to a function, defined between parentheses.

FUNCTION CALL: It is a statement which executes the function.

ARGUMENT: It is a value which is provided at the time of function calling. It is specified within parentheses.

RETURN VALUE: The value returned by the function as output to the caller.

MODULE: A file that contains a collection of related functions and definitions.

IMPORT STATEMENT: It is used to import various modules in Python

STRING A group of characters

SELF – ASSESSMENT QUESTIONS

1. What are the benefits of using functions? Also differentiate between function definition and function calling.
2. What is Scope of a variable? Explain local and global variables with example.
3. Define the return statement in a function. Give the syntax.
4. Discuss the built in methods and functions of string.
5. What are modules? How do you use them in your programs?

EXERCISES

1. Write a function to convert a decimal number to its binary, octal and hexadecimal equivalents.
2. Write a function to find the sum of several natural numbers using recursion
3. Write a program to declare string do the following:
    - Convert to lower case
    - Convert to upper case
    - Return the index of first ‘a’
    - Replace each ‘a’ by ‘p’
    - Count the number of ‘o’
    - Find the length
4. Write a program to define function to find the largest of given two numbers. Import that function in another program and pass values.
5. Write a program to demonstrate the use of dir().

QUIZ - ANSWERS

1. d.All
2. b.def
3. c.module
4. b.()
5. a.exit a function
6. string
7. True
8. True
9. False
10. False
11. b..py
12. from math import sqrt, cos
13. True
14. True
15. True

SUGGESTED READINGS

1. Yashavant Kanetkar and Aditya Kanetkar, “Let us Python Solutions”, bpb publications, 6th Edition, 2023

2. Ralph T.Burwell,” Fundamentals of Python: Basics of Python coding and step-by-step instructions for complete novices” Kindle Edition

3. David Amos, Dan Bader, Joanna Jablonski · “,Python Basics:A Practical Introduction to Python 3”, Real Python (Realpython.Com) Fourth Edition, 2021

OPEN SOURCE E-CONTENT LINKS

- https://www.python.org/about/gettingstarted/
- https://www.tutorialspoint.com/python/python_literals.htm
- https://www.w3schools.com/python/
- https://docs.python.org/3/tutorial/index.html
- https://www.geeksforgeeks.org/python-programming-language/

REFERENCES

1. Reema Thareja,""Python Pogramming using Problem Solving approach", Oxford Higher Education,
2. Dr. R. NageswaraRao, "Core Python Programming", First Edition, 2017, Dream tech Publishers.
3. VamsiKurama, "Python Programming: A Modern Approach", Pearson Education.
4. Mark Lutz, "Learning Python", Orielly.
5. E Balagurusamy , "Problem Solving and Python Programming", McGraw Hill Education (India) Private Limited,

PYTHON PROGRAMMING

UNIT 4- PYTHON PROGRAMMING

Lists: Creating a list -Access values in List-Updating values in Lists- Nested lists - Basic list operations- List Methods. Tuples: Creating, Accessing, Updating and Deleting Elements in a tuple – Nested tuples– Difference between lists and tuples. Dictionaries: Creating, Accessing, Updating and Deleting Elements in a Dictionary – Dictionary Functions and Methods - Difference between Lists and Dictionaries.

Functions, String, and Modules

UNIT OBJECTIVES

In this unit, learner will explore the various data types supported by Python which includes List, Tuple and Dictionary. How to create the above three? How to access the values in it? How to update or delete an element in it? The unit also discusses the different functions and methods that can be performed on List, Tuple and Dictionary. Finally the differences between List and Tuple, List and Dictionary are also discussed.

SECTION 4.1: LIST

A list is a sequence of items separated by commas and items enclosed in square brackets [ ]. These are similar to arrays available with other programming languages but unlike arrays items in the list may be of different data types. Lists are mutable, and hence, they can be altered even after their creation.

Lists in Python are ordered and have a definite sequence and the indexing of a list is done with 0 being the first index. Each element in the list has its definite

place in the list, which allows duplicating of elements in the list, with each element having its own distinct place and creditability. It is represented by list class.

4.1.1 – Creating a List

lst_var=[val1,val2..]

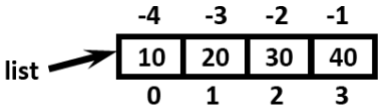
Example

>>>lst1[1,2,3,4,5] >>>print(lst1) [1,2,3,4,5]	>>>lst1=['a','b','c','d'] >>>print(lst2) ['a','b','c','d']
>>>lst3=["Python", "Programming"] >>>print(lst3) ["Python", "Programming"]	>>>lst4=['1','ball',9,34.89] >>>print(lst4) ['1','ball',9,34.89]

4.1.2 – Access values in a List

List can also be sliced and concatenated. To access values in a lists, square brackets are used to slice along with the index or indices to get value stored at that index.

seq=List[start:stop:step]



Example

numlst=[1,2,3,4,5,6,7,8,9,10]

print("numlist is :",numlst)

```
print("First element in numlist is :",numlst[0])
print("numlst[2:5] =",numlst[2:5])
print("numlst[::2] =",numlst[::2])
print("numlst[1::3] =",numlst[1::3])
```

Output:

```
numlist is : [1,2,3,4,5,6,7,8,9,10]
First element in numlist is :  1
numlst[2:5] =[3,4,5]
numlst[::2] = [1,3,5,7,9]
numlst[1::3] =[2,5,8]
```

4.1.3 - Updating values in List

Once created, one or more elements of a list can be easily updated by giving the slice on the left-hand side of the assignment operator. You can also append new values in the list and remove existing values from the list using the append( ) method and del statement respectively.

Example:

```
num_list= [1,2,3,4,5,6,7,8,9,10]
print("list is:",num_list)
num_list[5]=100
print("List after updation is:",num_list)
num_list.append(200)
print("List after appending a value is: ",num_list)
del num_list[3]
print("List after deleting a value is:",num_list)
```

Output:

list is: [1,2,3,4,5,6,7,8,9,10]

List after updation is: [1,2,3,4,5,100,7,8,9,10]

List after appending a value is: [1,2,3,4,5,100,7,8,9,10,200]

List after deleting a value is: [1,2,3,5,100,7,8,9,10,200]

4.1.4 - Nested List

Nested list means a list within another list.

Example

```
lst=[1.'a',"abc",[10,20,30],5.9]
I=0
while i<(len(lst)):
        print("List [",I,"] = ", lst[i])
        i+=1
print("Second element of nested list = ",lst[3][1])
```

OUTPUT:

List[0]=1

List[1]=a

List[2]=abc

List[3]=[10,20,30]

List[4]=5.9

Second element of nested list = 20

4.1.5 - Basic List operations

Operation	Description	Example	Output

len	Returns length of list	len([1,2,3,4,5,6,7,8,9,10])	10
concatenation	Joins two lists	[1,2,3,4,5]+[6,7,8,9,10]	[1,2,3,4,5,6,7,8,9,10]
repetition	Repeats elements in the lists	[“Hello”,”World”]*2	[“Hello”,”World”, “Hello”,”World”]
in	Checks if the value is present in the list	a" in[“a",“e",“i",“o",“u"]	True
not in	Checks if the value is not present in the list	3 not in [0,2,4,6,8]	True
max	Returns maximum value in the list	lst1=[6,3,7,0,1,2, 4,9] print(max(lst1))	9
min	Returns minimum value in the list	num_list=[6,3,7,0,1,2, 4,9] print(min(num_list))	0
sum	Adds the values in the list that has numbers	num_list=[1,2,3,4,5,6, 7,8,9,10] print(“SUM=”,sum(num_list))	SUM=55
all	Returns True if all elements of the list are true(or if the list is empty)	num_list=[0,1,2,3] print(all(num_list))	False
any	Returns True if any element of the list is true. if the list is empty return false	num_list=[6,3,7,0,1,2, 4,9] print(any(num_list))	True
list	converts iterable(tuple,string,set,dictionary)	list1=list(“HELLO”) print(list1)	[‘H’,’E’,’L’,’L’,’O’]
sorted	Returns a new sorted list. The original list not sorted	list1=[3,4,1,2,7,8] list2=sorted(list1) print(list2)	[1,2,3,4,7,8]

4.1.6 – List Methods

Method & Description
cmp(list1,list2) It compares the elements of both the lists, list1 and list2.
len(list) It returns the length of the string, i.e., the distance from starting element to last element.
max(list) It returns the item that has the maximum value in a list.
min(list) It returns the item that has the minimum value in a list.
list(seq) It converts a tuple into a list.
list.append(item) It adds the item to the end of the list.
list.count(item) It returns number of times the item occurs in the list.
list.extend(seq) It adds the elements of the sequence at the end of the list.
list.index(item) It returns the index number of the item. If item appears more than one time, it returns the lowest index number.
list.insert(index,item) It inserts the given item onto the given index number while the elements in the list take one right shift.
list.pop(item=list[-1]) It deletes and returns the last element of the list.
list.remove(item) It deletes the given item from the list.
list.reverse() It reverses the position (index number) of the items in the list.
list.sort([func]) It sorts the elements inside the list and uses compare function if provided

pop Operator If we know the index of the element that we want to delete, then we can use the pop operator.

Example

```
list = [10,20,30,40]

a = list.pop(2)

print list

print a
```

Output

```
[10,20,40]

30
```

del Operator The del operator deletes the value on the provided index, but it does not store the value for further use.

Example

```
list = ['w','x','y','z']

del list(1)

print list
```

Output

```
['w', 'y', 'z']
```

remove Operator We use the remove operator if we know the item that we want to remove or delete from the list (but not the index).

Example

```
list = [10,20,30,40]

 list.remove(10)

print list
```

Output

[20,30,40]

Let's Sum Up

In this section we have studied the various features of List in Python Programming Language. How to create a List, access the elements in it, how to delete or update the elements with the basic operations that can be performed in a List. Finally discussed the functions and methods that can be performed in a List.

Check your progress -QUIZ

1. Which of the following will separate all the items in list?

   a. * b. , c. ; d. & 19.

2. What is the repetition operator in lists?

   a. * b. , c. ; d. &

3. Which of the following functions will sort a list?

   a. list.sort b. list.sort([func]) c. list.sort[func] d. list.sort(func)

4. What will be the output of the given code? list = ['john', 'book', 123, 3.45, 105, 'good'] >>>print (list[4:])

   a. [3.45, 105, 'good'] b. ['john', 'book', 123, 3.45]

   c. [105, 'good'] d. [123, 3.45] 22.

5. Which of the following functions will give the total length of a list?

   a. Len b. len(list) c. max(len) d. max len(list)

SECTION 4.2: TUPLE

Like list, Tuple is another data structure supported by Python. It is very similar to list but differs in two things.

- First, a tuple is a sequence of **immutable** objects. This means you cannot change the values in a tuple.

- Second, tuples uses parentheses to define its elements where as list uses square brackets.

4.2.1 - Creating a Tuple

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses.

Syntax:

Tup1=(val1,val2,….)

where val (or values) can be an integer, a floating number, a character, or a string.

Examples

1) Tup1=( ) #creates an empty tuple.

print(Tup1)

Output:

Note: no output will be displayed.

2) Tup1=(5) #creates a tuple with single element

print(Tup1)

Output:

5

3) Tup1=(1,2,3,4,5) #creates a tuple of integers

print (Tup1)

Tup2=(„a",‟b",‟c",‟d") #creates a tuple of characters

print(Tup2)

Tup3=(“abc”,”def”,”ghi”) #creates a tuple of strings

```
print(Tup3)

Tup4=(1.2,2.3,3.4,4.5,5.6) #creates a tuple of floating point numbers

print(Tup4)

Tup5=(1,”abc”,2.3,"d") #creates a tuple of mixed values

print(Tup5)
```

Output:

```
1,2,3,4,5

“a","b","c","d"

“abc","def","ghi"

1.2,2.3,3.4,4.5,5.6

1,"abc",2.3,"d"
```

4) A Tuple with parenthesis

```
print(“a",”bcd”,2,4.6)
```

Output:

```
A bcd2 4.6
```

5) Default Tuple without parenthesis

```
a,b=10,20

print(a,b)
```

Output:

```
10 20
```

4.2.2 – Accessing Tuple

Like strings and lists tuples indices also starts with 0. The operations performed are slice, concatenate, etc.. To access values in tuple, slice operation is used along with the index

Example :

```
1)  Tup1=(1,2,3,4,5,6,7,8,9,10)
    print(“Tup[3:6]=”,Tup1[3:6])
    print(“Tup[:8]=”,Tup1[:4])
    print(“Tup[4:]=”,Tup1[4:])
    print(“Tup[:]=”,Tup1[:])
```

Output:

```
    Tup[3:6]=(4,5,6)
    Tup[:8]=(1,2,3,4)
    Tup[4:]=5,6,7,8,9,10)
    Tup[:]=(1,2,3,4,5,6,7,8,9,10)
```

The tuple values can be accessed using square brackets:

```
2)  tuple =(1,2,3,4,5.5,"str")
      print tuple
     print tuple[5]
     print tuple[1:5
```

OUTPUT:

```
    1,2,3,4,5.5,"str"
    "str"
    2,3,4,5.5
```

4.2.3 Updating Tuple

As we all know tuples are immutable objects so we cannot update the values but we can just extract the values from a tuple to form another tuple.

Example:

```
1)  Tup1=(1,2,3,4,5)
```

```
Tup2=(6,7,8,9,10)

Tup3=Tup1+Tup2

print(Tup3)
```

Output:

```
(1,2,3,4,5,6,7,8,9,10)
```

2)

```
Tup1=(1,2,3,4,5)

Tup2=(„sree",'vidya','ram')

Tup3=Tup1+Tup2

print Tup3
```

Output:

```
(1,2,3,4,5,'sree','vidya','ram')
```

4.2.4 Deleting element in Tuple

Deleting a single element in a tuple is not possible as we know tuple is a immutable object. Hence there is another option to delete a single element of a tuple i.e. you can create a new tuple that has all elements in your tuple except the ones you don't want.

Example

1)

```
Tup1=(1,2,3,4,5)

del Tup1[3]

print Tup1
```

Output:

```
Traceback (most recent call last):
File "test.py", line 9, in <module>
del Tup1[3]
Type error: “tuple‟ object doesn’t support item deletion
```

2)**delete the entire tuple by using del statement**.

```
Tup1=(1,2,3,4,5)
del Tup1
print Tup1
```

Output:

```
Traceback (most recent call last):
File "test.py", line 9, in <module>
print Tup1;
NameError: name 'Tup1' is not defined
```

Note: Note that exception is raised because you are now trying to print a tuple that has already been deleted.

Basic tuple operations:

Like strings and lists, you can also perform operations like concatenation, repetition, etc. on tuples. The only difference is that a new tuple should be created when a change is required in an existing tuple.

Operation	Expression	Output
Length	len((1,2,3,4,5,6))	6
Concatenation	(1,2,3)+(4,5,6)	(1,2,3,4,5,6)

Repetition	(“Good..”)*3	Good.. Good.. Good..
Membership	5 in (1,2,3,4,5,6,7,8,9)	True
Iteration	for i in (1,2,3,4,5,6,7,8,9,10): print(i,end=” “)	1,2,3,4,5,6,7,8,9,10
Comparision(Use >,<,==)	Tup1=(1,2,3,4,5) Tup2=(1,2,3,4,5) print(Tup1>Tup2)	True
Maximum	max(1,3,5,7,9,0)	9
Minimum	min(1,3,5,7,9,0)	0
Convert to tuple(converts a sequence into a tuple)	tuple(“Hello”) tuple([1,2,3,4,5])	(„H",”e",”l",”l",”o") (1,2,3,4,5)
Sorting(The sorted( ) function takes elements in a tuple and returns a new sorted list (does not sort the tuple itself)).	t=(4,56,-9) sorted(t)	-9,4,56

4.2.5 Nested Tuple

Python allows you to define a tuple inside another tuple. This is called a *nested tuple.*

Example

```
stud=((“Anu”,”Bsc”,89.0),(“Banu”,”BA”,78.0),(”Chandru”,”BCom”,95.0))

for I in stud:
```

```
print(i)
```

Output

```
("Anu","Bsc",89.0)

("Banu","BA",78.0)

("Chandru","BCom",95.0)
```

Example :- You can even specify a list within the tuple.

```
stud=(("Anu",[93,89,90,91])

print("Marks of ",stud[0], "is ..",stud[1:])
```

Output

```
Marks of Anu is ..[93,89,90,91]
```

4.2.6 – Difference between List and Tuple

1. List is mutable and tuple is non-mutable.
2. List elements are represented by using square brackets. Tuple elements are represented by using parenthesis.
3. To store tuple elements, Python Virtual Memory requires less memory. To store list elements, Python Virtual Memory requires more memory.
4. Tuple elements can be access within less time, because they are fixed (Performance is more). Performance is less compared with tuples.

Let's Sum Up

In this section, the learner studied the various features Tuples in Python Programming Language and how to create a tuple, access the elements in it? , how to delete or update the elements with the basic operations that can be performed in a tuple. Finally the section discussed the difference between a List and Tuple.

Check your progress - QUIZ

6. Which of the following sequence data type is similar to the tuple?
   a. Dictionaries b. List c. String d. function
7. In which operator are tuples enclosed?
   a. { } b. [ ] c. <> d. ( )
8. Which of the following is a Python tuple?
   a. [7, 8, 9] b. {7, 8, 9} c. (7, 8, 9) d. <7,8,9>
9. What will be the output of the following code? >>>tuple = (‘john’, 100, 345, 1.67, ‘book’) >>>print(tuple[0])
   a. John b. 0 c. Book d. error
10. Which of the following will not be correct if tuple = (10, 12, 14, 16, 18)?
    a. print(min(tuple)) b. print(max(tuple)) c. tuple[4] = 20 d. print(len(tuple)

SECTION 4.3: DICTIONARY

It is a data structure in which we store values as a pair of key and value. o Each key is separated from its value by a colon (:), and consecutive items are separated by commas. o The entire items in a dictionary are enclosed in curly brackets ({})

Syntax:

dictionary_name = {key_1: value_1, key_2: value_2, key_3: value_3}

If there are many keys and values in dictionaries, then we can also write just one key-value pair on a line to make the code easier to read and understand.

dictionary_name = {key_1: value_1, key_2: value_2, key_3: value_3 , ….}

- Keys in the dictionary must be unique and be of any immutable data type (like Strings, numbers, or tuples), there is no strict requirement for uniqueness and type of values.
- Values of a key can be of any type.
- Dictionaries are not Sequences, rather they are mappings.

- Mappings are collections of objects that are store objects by key instead of by relative position.

4.3.1– Creating Dictionary

A dictionary can be also created by specifying key-value pairs separated by a colon in curly brackets as shown below. One key value pair is separated from the other using a comma.

Example:

```
d= {'roll_no':'18/001','Name:':'Arav','Course':'B.tech'}

print(d)
```

Output:

```
{'Name:': 'Arav', 'Course': 'B.tech', 'roll_no': '18/001'}
```

4.3.2 – Access Dictionary

In Dictionary, through key accessing values,

Example:

```
d={'Name': 'Arav', 'Course': 'B.tech', 'roll_no': '18/001'}

print('d[Name]:',d['Name'])

print('d[course]:',d['Course'])

print('d[roll_no]:',d['roll_no'])
```

Output:

```
d[Name]: Arav

d[course]: B.tech

d[roll_no]: 18/001
```

4.3.3– Update Dictionary

To add a new entry or a key-value pair in a dictionary, just specify the keyvalue pair as you had done for the existing pairs.

Syntax

dictionary_ variable[key]= val

Example:

```
d={'Name': 'Arav', 'Course': 'B.tech', 'roll_no': '18/001'}
d['marks']=99 #new entry
print('d[Name]:',d['Name'])
print('d[course]:',d['Course'])
print('d[roll_no]:',d['roll_no'])
print('d[marks]:',d['marks'])
```

Output:

```
d[Name]: Arav
d[course]: B.tech
d[roll_no]: 18/001
d[marks]: 99
```

Example: **To modify an entry, just overwrite the existing value**

```
d={'Name': 'Arav', 'Course': 'B.tech', 'roll_no': '18/001'}
d['marks']=99 #new entry
print('d[Name]:',d['Name'])
```

```
print('d[course]:',d['Course'])

print('d[roll_no]:',d['roll_no'])

print('d[marks]:',d['marks'])

d['Course']='BCA' #Updated entry

print('d[course]:',d['Course'])
```

OUTPUT:

```
d[Name]: Arav

d[course]: B.tech

d[roll_no]: 18/001

d[marks]: 99

d[course]: BCA
```

4.3.4 – Deleting elements in Dictionary

You can delete one or more items using the del keyword. To delete or remove all the items in just one statement, use the clear () function. Finally, to remove an entire dictionary from the memory, we can gain use the del statement as del Dict_name.

syntax

del dictionary_variable[key]

Example:

```
dict_cubes = {1:1, 2:8, 3:9, 4:64, 5:125, 6:216}

print("remove a particular item ")

dict_cubes.pop(3)

print("Dictionary item:", dict_cubes)
```

```
print(“remove an arbitrary item”
dict_cubes.popitem()
print(“remove an arbitrary item”
dict_cubes.popitem()
print(“Dictionary item:”, dict_cubes)
print(“remove a particular item “)
deldict_cubes[6] # delete a particular item
print(“Dictionary item: “,dict_cubes)
dict_cubes.clear() # remove all items
dict_cubes
del dict_cubes # delete the dictionary itself
 print (dict_cubes)
```

Output

```
remove a particular item
9
Dictionary item: {1: 1, 2: 8, 4: 64, 5: 125, 6: 216}
remove an arbitrary item
(1, 1)
remove an arbitrary item
(2, 8)
Dictionary item: {4: 64, 5: 125, 6: 216}
{4: 64, 5: 125}
{}
Traceback (most recent call last):
File “<pyshell#40>”, line 1, in <module>
```

```
printdict_cubes
NameError: name 'dict_cubes' is not define
```

4.3.5 – Dictionary Functions and Methods

- **Traversing** Traversing in dictionary is done on the basis of keys. For this, for loop is used, which iterates over the keys in the dictionary and prints the corresponding values using keys.

Example

We will define a function print_dict. Whenever a dictionary is passed as an argument to this function, it will print the keys and values of the dictionary.

```
def print_dict(d):
    for c in d:
        print c,d[c]
dict1 = {1:'a',2:'b',3:'c',4:'d'}
print_dict(dict1)
```

OUTPUT:

```
1 a
2 b
3 c
4 d
```

- **Membership** Using the membership operator (in and not in), we can test whether a key is in the dictionary or not. We have seen the in operator earlier as well in the list and the tuple. It takes an input key and finds the key in the dictionary. If the key is found, then it returns True, otherwise, False.

Example

```
cubes = {1:1, 2:8, 3:27, 4:64, 5:125, 6:216}

print(3 in cubes)

print(7 not in cubes)

print(10 in cubes)
```

Output:

```
True

True

False
```

Built-In Dictionary Methods

Function & Description

1. **all(dict)** It is a Boolean type function, which returns True if all keys of dictionary are true (or the dictionary is empty).
2. **any(dict)** It is also a Boolean type function, which returns True if any key of the dictionary is true. It returns false if the dictionary is empty.
3. **len(dict)** It returns the number of items (length) in the dictionary.
4. **cmp(dict1,dict2)** It compares the items of two dictionaries.
5. **sorted(dict)** It returns the sorted list of keys.
6. **str(dict)** It produces a printable string representation of the dictionary.
7. **dict.clear()** It deletes all the items in a dictionary at once.
8. **dict.copy()** It returns a copy of the dictionary.
9. **dict.fromkeys()** It creates a new dictionary with keys from sequence and values set to value.
10. **dict.get(key, default=None)** For key key, returns value or default if key not in dictionary.
11. **dict.has_key(key)** It finds the key in dictionary; returns True if found and false otherwise.
12. **dict.items()** It returns a list of entire key: value pair of dictionary.

13. **dict.keys()** It returns the list of all the keys in dictionary.
14. **dict.setdefault (key, default=None)** Similar to get(), but will set dict[key]=default if key is not already in dict.
15. **dict.update(dict2)** It adds the items from dict2 to dict.
16. **dict.values()** It returns all the values in the dictionary

4.3.6 – Difference between List and Dictionary

These are the main differences between a list and a dictionary.

- First, a list is an ordered set of items. But, a dictionary is a data structure that is used for matching one item (key) with another (value).
- Second, in lists, you can use indexing to access a particular item. But, these indexes should be a number. In dictionaries, you can use any type(immutable) of value as an index.
- Third, lists are used to look up a value whereas a dictionary is used to take one value and look up another value. For this reason, dictionary is also known as a *lookup table.*
- Fourth, the key-Value pair may not be displayed in the order in which it was specified while defining the dictionary. This is because Python uses complex algorithms(called hashing) to provide fast access to the items stored in the dictionary. This also makes dictionary preferable to use over a list.

Let's Sum Up

In this section we have studied the various features Dictionary in Python Programming Language. How to create a Dictionary, access the elements in it, how to delete or update the elements with the basic operations that can be performed in a Dictionary. Finally discussed the difference between a List and Dictionary.

.

Check your progress - QUIZ

11. Which core data type in Python is an unordered collection of key-value pairs?
    a. Tuple b. dictionary c. function d. list

12. In which of the following operators are dictionaries enclosed?
    a. { } b. ( ) c. [ ] d. <>
13. Which of the following represent keys in the dictionary?
    a. function or list b. numbers or list
    c. strings or functions d. numbers or strings
14. Which operator is used to access the values in dictionary?
    a. { } b. ( ) c. [ ] d. <>
15. Which of the following forms do dictionaries appear in?

a. keys and values b. only keys c. list and keys d. only values

UNIT SUMMARY

In this unit, the learner had learned about list which is a collection of items or elements; the sequence of data in a list is ordered. The elements or items in a list can be accessed by their positions, i.e., indices. Lists are mutable which means that we can change the value of any element inside the list at any point of time. Tuples are the sequences of different types of values. Tuples are immutable and thus the elements or values cannot be modified. A dictionary is a mapping between some set of keys and values. Each key is associated with a value. The mapping of a key and value is called a key-value pair and together they form one item or element. The values in a dictionary are not unique and can be duplicated, but the keys in the dictionary are unique. The difference between the accessing methods of dictionary is that when the key is not found in dictionary, it returns none instead of KeyError. Dictionaries are mutable and thus the elements or values can be modified.

GLOSSARY

LIST: It is a series or a sequence of different data items.

ELEMENT: An element is a value in the list, also called item.

INDEX: It is an integer value that indicates the position of an element in a list.

LIST TRAVERSAL: Accessing all the items in a list.

OBJECT: It is something a variable can refer to. An object has a type and value.

EQUIVALENT: It means having equal values.

IDENTICAL: It means same objects (which implies equivalence).

TUPLE: Tuples, just like lists, are the sequence or series of different types of values that are separated by commas (,).

TUPLE ASSIGNMENT: It allows assignment of values to a tuple of variables on the left side of assignment from the tuple of values on the right side of the assignment.

VARIABLE-LENGTH ARGUMENT TUPLES: A variable number of arguments can also be passed to a function. A variable name which is preceded by an asterisk (*) collects the arguments into a tuple.

CONCATENATION: This operator works in tuples in the same way as in lists. This operator concatenates two tuples. This is done by the + operator in Python.

REPETITION: This operator repeats the tuples a given number of times. Repetition is performed by * operator.

in OPERATOR: This operator tells the user whether the given element exists in the tuple or not. It gives a Boolean output, i.e., TRUE or FALSE.

ITERATION: Iteration can be done in tuples using for loop. It helps in traversing the tuple.

len(tuple): It returns the length of the tuple.

DICTIONARY: The Python dictionary is an unordered collection of items or elements. The dictionary has a key: value pair.

KEY: It is used to get the value in the dictionary.

KEY-VALUE: This pair represents the items in the dictionary

SELF – ASSESSMENT QUESTIONS

1. Explain list with its syntax.

2. List out the methods in list with example

3. What is a tuple and how is it created in Python?
4. How are the values in a tuple accessed?
5. What are the different methods used in deleting elements from dictionary?

EXERCISES

1. How can we add an element in the list? Write a program to insert 32 to the fourth position in the given list [1, 4, 23, 56, 90].
2. Write a program to reverse a list.
3. Consider the tuple (1,3,5,7,9,2,4,6,8,10). Write a program to print half its values in one line and the other half in the next line.
4. Write a python program to demonstrate tuples functions and operations
5. Write a python program to demonstrate dictionary functions and operations

QUIZ - ANSWERS

1. b,
2. a.*
3. b. list.**sort**([func])
4. [105,'good']
5. b.len(list)
6. b.list
7. d.()
8. (7,8,9)
9. a.john
10. c. tuple[4] = 20
11. b.dictionary
12. a.{}
13. d.numbers or strings
14. c.[]
15. a.keys and values

SUGGESTED READINGS

1. Yashavant Kanetkar and Aditya Kanetkar, “Let us Python Solutions”, -bpb publications, 6th Edition, 2023

2. Ralph T.Burwell,” Fundamentals of Python: Basics of Python coding and step-by-step instructions for complete novices” Kindle Edition

3. David Amos, Dan Bader, Joanna Jablonski “,PythonBasics:A Practical Introduction to Python 3”, Real Python (Realpython.Com) Fourth Edition, 2021

OPEN SOURCE E-CONTENT LINKS

- https://www.python.org/about/gettingstarted/
- https://www.guru99.com/python-tutorials.html
- https://www.w3schools.com/python/
- https://docs.python.org/3/tutorial/index.html
- https://www.geeksforgeeks.org/python-programming-language/

REFERENCES

1. Reema Thareja,”Python Pogramming using Problem Solving approach”, Oxford Higher Education,

2. Fabio Nelli, Python Data Analytics”, APress.

3. VamsiKurama, “Python Programming: A Modern Approach”, Pearson Education.

4. Mark Lutz, “Learning Python”, Orielly.

5. E Balagurusamy , “Problem Solving and Python Programming”, McGraw Hill Education (India) Private Limited,

PYTHON PROGRAMMING

UNIT 5 - PYTHON PROGRAMMING

Python File Handling: Types of files in Python - Opening and Closing files-Reading and Writing files: write() and writelines() methods- append() method – read() and readlines() methods – with keyword – Splitting words – File methods - File Positions- Renaming and deleting files

Python File Handling

UNIT OBJECTIVES

In this unit, learners will have a elaborate understanding of File concepts, various file types in Python. How to write the contents into a file using write or writeline method. Similarly how to read the contents from the file using read or readline method. Before doing any operations in a file it has to be opened which has various modes like read, write, append, binary or text, etc., Once the job gets completed it has to be closed. While reading a file user can navigate to a particular location in a file. The way to rename or delete a file is also discussed.

SECTION 5.1: PYTHON FILE HANDLING

A file is a collection of data stored on a secondary storage device like hard disk. When a program is being executed, its data stored in *random access memory* (RAM). Though RAM can be accessed faster by the CPU, it is **volatile**, which means when the program ends, or the computer shuts down, all the data is lost. If you want the data for future use, then it has to be stored in a permanent or non-volatile storage media.

Data on non-volatile storage media is stored in named locations on the media called **files.** To work with files first we must open it, read the contents that you previously written in it or write some new content into it. After using the file it should be closed.

A file is basically used because real life applications involve large amount of data and in such situations console oriented I/O operations pose two major problems:

- First, it becomes cumbersome and time consuming to handle large amount of data through terminals.
- Second, when doing I/O using terminal, the entire data is lost when either the program is terminated or computer is turned off. Therefore, it becomes

necessary to store data on a permanent storage(the disks) and read whenever necessary, without destroying the data.

5.1.1 – Types of Files in Python

Python supports two types of files

- ASCII Text Files
- Binary Files

ASCII Text Files

A text file is a stream of characters that can be sequentially processed by a computer in forward direction. For this reason a text file is usually opened for only one kind of operation reading, writing, or appending) at any given time. Because text files can process characters, they can only read or write data one character at a time. In Python, a text stream is treated as a special kind of file. Depending on the requirements of the operating system and on the operation that has to be performed (read/write operation) on the file, the newline characters may be converted to or from carriage-return/linefeed combinations. Besides this, other character conversions may also be done to satisfy the storage requirements of the operating system. However, these conversions occur transparently to process a text file. In a text file, each line contains zero or more characters and ends with one or more characters.

Another important thing is that when a text file is used, there are actually two representations of data- internal or external. For example, an integer value will be represented as a number that occupies 2 or 4 bytes of memory internally but externally the integer value will be represented as a string of characters representing its decimal or hexadecimal value.

***Note**: In a text file, each line of data ends with a newline character. Each file ends with a special character called end-of-file (EOF) Marker.*

Binary Files

A binary file is a file which may contain any type of data, encoded in binary form for computer storage and processing purposes. It includes files such as word

processing documents, PDFs, images, spreadsheets, videos, zip files and other executable programs. Like a text file, a binary file is a collection of bytes. A binary file is also referred to as a character stream with following two essential differences.

A binary file does not require any special processing of the data and each byte of data is transferred to or from the disk unprocessed. Python places no constructs on the file, and it may be read from, or written to, in any manner the programmer wants. While text files can be processed sequentially, binary files, on the other hand, can be either processed sequentially or randomly depending on the needs of the application.

***Note**: Binary files store data in the internal representation format. Therefore, an integer value will be stored in binary form as 2 byte value. The same format is used to store data in memory as well as in files. Like Text files, Binary files also ends with an EOF.*

When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed. Python has many in-built functions and methods to manipulate files. Hence, in Python, a file operation takes place in the following order

1. Open a file
2. Read or write (perform operation)
3. Close the file

5.1.2 – Opening and Closing Files

Before reading from or writing to a file, you must first open it using Python's built-in open() function. This function creates a file object, which will be used to invoke methods associated with it.

Syntax

fileObj = open(file_name [, access_mode])

- file_name is a string value that specifies name of the file that you want to access
- access_mode indicates the mode in which the file has to be opened, i.e. read, write, append, etc. Access mode is an optional parameter and the default file access mode is read(r).

EXAMPLE

```
file=open("file1.txt","rb")

print(file)
```

OUTPUT

```
<open file 'file1.txt',mode 'rb' at 0x02A850D0>
```

Access modes

Python supports the following access modes for opening a file those are:

Modes	Description
r	It opens a file in reading mode. The file pointer is placed at the starting of the file. It is the default mode.
rb	It opens a file in reading only mode in binary format. The file pointer is placed at the starting of the file.
r+	It opens the file in both reading and writing mode. The file pointer is placed at the starting of the file.
rb+	It opens the file in both reading and writing mode in binary format. The file pointer is placed at the starting of the file.
W	It opens the file in writing only mode. If a file exists, it overwrites the existing file; otherwise, it creates a new file.
Wb	It opens the file in writing only mode in binary format. If a file exists, it overwrites the existing file; otherwise, it creates a new file.
w+	It opens the file in booth reading and writing mode. If a file exists, it overwrites the existing file; otherwise, it creates a new file.
wb+	It opens the file in booth reading and writing mode in binary format. If a file exists, it overwrites the existing file; otherwise, it creates a new file.
a	It opens a file for appending. The file pointer is placed at the end of the file. If the file does not exist in the directory, it creates a new file for writing.
ab	It opens a file for appending in binary format. The file pointer is placed at the end of the file. If the file does not exist in the directory, it creates a new file for writing.
a+	It opens a file for appending and reading. The file pointer is placed at the end of the file. If the file does not exist in the directory, it creates a new file for writing.
ab+	It opens a file for appending and reading in binary format. The file pointer is placed at the end of the file. If the file does not exist in the directory, it creates a new file for writing.

The File Object Attributes

Once a file is successfully opened, a file object is returned. Using this file object, you can easily access different type of information related to that file. This information can be obtained by reading values of specific attributes of the file. The Following table shows list attributes related to file object.

- **fileobj.closed –** Returns True is file is closed and False otherwise
- **fileobj.mode –** Returns access mode with which file has been opened
- **fileobj.name –** Retrurns the name of the file

Example

```
# open a file
f = open("C:/Python27/test.txt","wb")
print (f.name)
print (f.closed)
print (f.mode)
print (f.softspace)
f.close() # Close the opened file
 print (f.closed)
```

Output :

```
C:/Python27/test.txt
False
Wb
O
True
```

Closing A File

The close() method is used to close the file object. Once a file object is closed, you cannot further read from or write into the file associated with the file object. While closing the file object the close() flushes any unwritten information. Although, Python automatically closes a file when the reference object of a file is reassigned to another file, but as a good programming habit you should always explicitly use the close() method to close a file.

Syntax:

fileObj.close()

The close() method frees up any system resources such as file descriptors, file locks, etc. that are associated with the file. Once the file is closed using the close() method, any attempt to use the file object will result in an error.

Example: Write a Python program to assess if a file is closed or not..

```
file = open('File1.txt','wb')
print('Name of the file :',file.name)
print('File is closed:',file.closed)
print('File is now being closed')
file.close()
print('File is closed',file.closed)
print(file.read())
```

Output:

```
Name of the file : File1.txt
File is closed: False
File is now being closed
File is closed True
```

```
Traceback (most recent call last):
 File "D:/Python/sample.py", line 7, in <module>
 print(file.read())
io.UnsupportedOperation: read
```

5.1.3 – Reading and Writing Files

The read() and write() are used to read data from file and write data to file respectively. Both functions are used to manipulate data through files.

5.1.3.1 – write() and writelines() methods

The write() method is used to write a string to an already opened file. Of course this string may include numbers, special characters or other symbols.While writing data to a file, you must remember that the write() method does not add a newline character ('\n') to the end of the string.

Syntax

fileObj.write(string)

Example:Program that writes a message in the file,data.txt

```
file=open('data.txt','w')
file.write('hello cse we are learning python programming')
file.close()
print('file writing successful')
```

Output:

```
file writing successful
```

writeline() method:

The writelines() method is used to write a list of strings.

Example: Program to write to a file using the writelines() method

```
file=open('data.txt','w')
lines=['hellocse',' hope to enjoy',' learning','python programming']
file.writelines(lines)
file.close()
print('file writing successful')
```

Output:

```
file writing successful
```

5.1.3.2 – append() method

Once you have stored some data in a file,you can always open that file again to write more data or append data to it. To append a file, you must open it using „a" or „ab" mode depending on whether it is text file or binary file. Note that if you open a file with „w" or „wb" mode and then start writing data into it, then the existing contents would be overwritten.

Example:Program to append data to an already existing file

```
file=open('data.txt','a')
file.write('\nHave a nice day')
file.close()
print('Data appended successful')
```

Output:

```
Data appended successful
```

data.txt

hellocsehope to enjoylearning python programming
Have a nice day

5.1.3.3 – read() and readlines() methods

The read() method is used to read a string from an already opened file. As said before, the string can include, alphabets, numbers, characters or other symbols.

Syntax

fileObj.read([count])

- count is an optional parameter which if passed to the read() method specifies the number of bytes to be read from the opened file.

The read() method starts reading from the beginning of the file and if count is missing or has a negative value then, it reads the entire contents of the file (i.e., till the end of file).

Example1:Program to print the first 8 characters of the file data.txt

```
file=open('data.txt','r')
print(file.read(8))
file.close()
```

Output:

```
hellocse
```

Example2:Program to display the content of file using for loop

```
file=open('data.txt','r')
for line in file:
 print(line)
```

```
file.close()
```

Output:

```
hellocsehope to enjoylearningpython programming
Have a nice day
```

read() methods returns a newline as “\n"

readline() Method

It is used to read single line from the file. This method returns an empty string when end of the file has been reached.

Example Program to demonstrate the usage of readline() function

```
file=open('data.txt','r')
print('firtsline:',file.readline())
print('second line:',file.readline())
print('third line:',file.readline())
file.close()
```

Output:

```
firtsline: hellocsehope to enjoylearningpython programming
second line: Have a nice day
third line:
```

readlines() Method

readlines() Method is used to read all the lines in the file.

Example:Program to demonstrate readlines() function

```
file=open('data.txt','r')
```

```
print(file.readlines())

file.close()
```

Output:

```
['hellocsehope to enjoylearningpython programming\n', 'Have a nice day']
```

list() Method

list() method is also used to display entire contents of the file. you need to pass the file object as an argument to the list() method.

Example: display the contents of the file data.txt using the list() method

```
file=open('data.txt','r')

print(list(file))

file.close()
```

Output:

```
['hellocsehope to enjoylearningpython programming\n', 'Have a nice day']
```

5.1.3.4 – with keyword

It is good programming habit to use the with keyword when working with file objects. This has the advantage that the file is properly closed after it is used even if an error occurs during read or write operation or even when you forget to explicitly close the file. When you open a file for reading, or writing, the file is searched in the current directory. If the file exists somewhere else then you need to specify the path of the file.

Example:

```
with open("file1.txt", "rb") as file:
    for line in file:
        print(line)
print("Let's check if the file is
closed : ", file.close())
```

OUTPUT

```
Hello World
Welcome to the world of Python
Programming.
Let's check if the file is closed :  True
```

```
file = open("file1.txt", "rb")
for line in file:
      print(line)
print("Let's check if the file is
closed : ", file.close())
```

OUTPUT

```
Hello World
Welcome to the world of Python
Programming.
Let's check if the file is closed : False
```

5.1.3.5 – splitting words

Python allows you to read line(s) from a file and splits the line (treated as a string) based on a character. By default, this character is space but you can even specify any other character to split words in the string.

Example: Program to split the line into series of words and use space to perform the split operation

```
with open('data.txt','r') as file:
    line=file.readline()
    words=line.split()
    print(words)
```

Output:

['hellocsehope', 'to', 'enjoylearningpython', 'programming']

5.1.3.6 – File methods

Method	Description	Example

`fileno()`	Returns the file number of the file (which is an integer descriptor)	`file = open("File1.txt", "w")`
		`print(file.fileno())`
		OUTPUT
		`3`

`flush()`	Flushes the write buffer of the file stream	`file = open("File1.txt", "w")` `file.flush()`

`isatty()`	Returns True if the file stream is interactive and False otherwise	`file = open("File1.txt", "w")`
		`file.write("Hello")`
		`print(file.isatty())`
		OUTPUT
		`False`

`readline(n)`	Reads and returns one line from file. n is optional. If n is specified then atmost n bytes are read	`file = open("Try.py", "r")`
		`print(file.readline(10))`
		OUTPUT
		`file = ope`

`truncate(n)`	Resizes the file to n bytes	`file = open("File.txt", "w")`
		`file.write("Welcome to the world of programming....")`
		`file.truncate(5)`
		`file = open("File.txt", "r")`
		`print(file.read())`
		OUTPUT
		`Welco`

`rstrip()`	Strips off whitespaces including newline characters from the right side of the string read from the file.	`file = open("File.txt")`
		`line = file.readline()`
		`print(line.rstrip())`
		OUTPUT
		`Greetings to All !!!`

5.1.4 – File Position

With every file, the file management system associates a pointer often known as file pointer that facilitates the movement across the file for reading and/ or writing data. The file pointer specifies a location from where the current read or write operation is initiated. Once the read/write operation is completed, the pointer is automatically updated. Python has various methods that tells or sets the position of the file pointer.

For example, the **tell()** method tells the current position within the file at which the next read or write operation will occur. It is specified as number of bytes from the beginning of the file. When you just open a file for reading, the file pointer is positioned at location¬ 0, which is the beginning of the file.

Syntax

seek(offset[, from])

- offset argument indicates the number of bytes to be moved
- from argument specifies the reference position from where the bytes are to be moved.

Example: Program that tells and sets the position of file pointer

File1.txt

Hello All,

Hope you are enjoying learning python

```
file=open("File1.txt","rb")
print("Position of file pointer before reading is :",file.tell())
print(file.read(10)
print("Position of file pointer after reading is :",file.tell())
print("Setting 3 bytes from the current position of file pointer")
file.seek(3,1)
print(file.read())
file.close()
```

OUTPUT:

```
Position of file pointer before reading is :0
Hello All,
Position of file pointer after reading is : 10
Setting 3 bytes from the current position of file pointer
pe you are enjoying learning python
```

5.1.5 – Renaming and Deleting Files

The os module in Python has various methods that can be used to perform fileprocessing operations like renaming and deleting files. To use the methods defined in the os module, you should first import it in your program then call any related functions.

- **rename() Method:** The **rename**() method takes two arguments, the current filename and the new filename.

  Syntax

  os.rename(old_file_name, new_file_name)

- **remove() Method**: This method can be used to delete file(s). The method takes a filename (name of the file to be deleted) as an argument and deletes that file.

  Syntax

  os.remove(file_name)

Example: Program to rename file 'File1.txt' to 'student.txt'

```
import os
os.rename("File1.txt","students.txt")
print("File Renamed")
```

Output:

```
File Renamed
```

Example Program to delete a file named File1.txt

```
import os
os.remove("File1.txt")
print("File Deleted")
```

Output:

```
File Deleted
```

Let's Sum Up

In this section, the user learned the concept of File. How to read a text file character by character or line by line. The text content can be written into text file using write() or writeline() method. Although file I/O operations is almost same as I/O, the only difference is that when doing file I/O, the user must specify the name of the file from which data should be read / written along with mode. The default mode is read. Using append() the content can be added to the existing file. The split() method used to split the words in a file. The file pointer can be moved to a particular location using seek() function and the current location in a file can be tracked using tell() method.

Check your progress

1. Which of the following functions is used to open a file in Python?

   a. open{} b. open() c. open[] d. Open()

2. What is a file object also known as?

   a. Object source b. File c. Object file d. Handle

3. Which of the following is the default mode while opening a file?

   a. Binary b. Number c. Text d. None

4. What is the syntax for opening a file in current directory?

   a. f = open("xyz.txt",'r') b. f = open("xyz.txt,'w')

   c. f = open("xyz.txt,'a') d. f = open("xyz.txt")

5. At what position will the file pointer be placed whenever we open a file for reading or writing?

   a. Middle b. Beginning c. Second line d. End

6. What is the default file access mode?

   a. write (w) b. append c. read (r) d. None

7. Whenever we open a file for appending, at what position will the file pointer be placed?

a. Middle b. Beginning c. Second line d. End

8. Which of the following statements is not correct?

a. When a file is opened for reading, if the file does not exist, an empty file will be opened.

b. When a file is opened for writing, overwrites the file if the file exists.

c. When a file is opened for writing, if the file does not exist, creates a new file for writing.

d. When a file is opened for reading, if the file does not exist, an error occurs.

9. Which of the following statements is correct while using 'a+' mode for opening a file?

a. Opens a file for appending only

b. Opens a file for writing only

c. Opens a file for both appending and reading

d. Opens a file for both appending and writing

10. What is the syntax to close a file?

a. file.close() b. close() c. close(); d. fileObject.close()

UNIT SUMMARY

A Function is a block of code that performs a specific task. In this unit, we have discussed various aspects of functions such as how to create functions, their scope, passing arguments to function. In addition to these topics, file handling operations are also discussed in details such as how to interact with file, copying, deleting, etc.

GLOSSARY

- **Delimiter** one or more characters used to specify the boundary between different parts of text.
- **File** A stream of information that is usually stored on a permanent storage media like hard drive, floppy disk, CD-ROM, etc.
- **File handle** An object that allows you to manipulate / read/ write/ close the file.
- **File Path** A sequence of directory names that specifics the exact location of a file.
- **Non-volatile memory** Memory that can store data even when the power supply to the computer system is switched off.
- **Text file** A file having printable characters organized into lines separated by newline characters.
- **Volatile memory** Memory that loses data as soon as the computer system is switched off. RAM is an example of volatile memory.

SELF – ASSESSMENT QUESTIONS

1. What are the different access modes in which you can open a file?
2. With example explain any three attributes of file object.
3. Is it mandatory to call close() method after using a file ?
4. Explain the syntax of read method.
5. How will you rename and delete a file in python?

EXERCISES

1. Write a program that tells and set the position of the file pointer.
2. Write a program that accepts filename as input, open the file and count the number of times a character appears in the file.

3. Write a program that reads data from a file and calculates the percentage of vowels and consonants in the file.

4. Write a program that reads a file and prints only those lines that has the word 'print'.

5. Write a program that has several lines. Each line begins with a line number. Read this file line by line and copy the line in another file but do not copy the numbers.

ANSWERS

1. b. open()
2. d. Handle
3. c. Text
4. a. f = open("xyz.txt",'r')
5. b. Beginning
6. c. read (r)
7. b. Beginning
8. a. When a file is opened for reading, if the file does not exist, an empty file will be opened
9. c. Opens a file for both appending and reading
10. d. fileObject.close()

SUGGESTED READINGS

1. Yashavant Kanetkar and Aditya Kanetkar, "Let us Python Solutions", -bpb publications, 6th Edition, 2023

2. Ralph T.Burwell," Fundamentals of Python: Basics of Python coding and step-by-step instructions for complete novices" Kindle Edition

3. David Amos, Dan Bader, Joanna Jablonski · ",PythonBasics:A Practical Introduction to Python 3", Real Python (Realpython.Com) Fourth Edition, 2021

OPEN SOURCE E-CONTENT LINKS

- https://www.python.org/about/gettingstarted/

- https://www.w3schools.com/python/
- https://docs.python.org/3/tutorial/index.html
- https://www.geeksforgeeks.org/python-programming-language/

REFERENCES

1. Reema Thareja,""Python Pogramming using Problem Solving approach", Oxford Higher Education,
2. Dr. R. NageswaraRao, "Core Python Programming", First Edition, 2017, Dream tech Publishers.
3. VamsiKurama, "Python Programming: A Modern Approach", Pearson Education.
4. Mark Lutz, "Learning Python", Orielly.
5. E Balagurusamy , "Problem Solving and Python Programming", McGraw Hill Education (India) Private Limited,