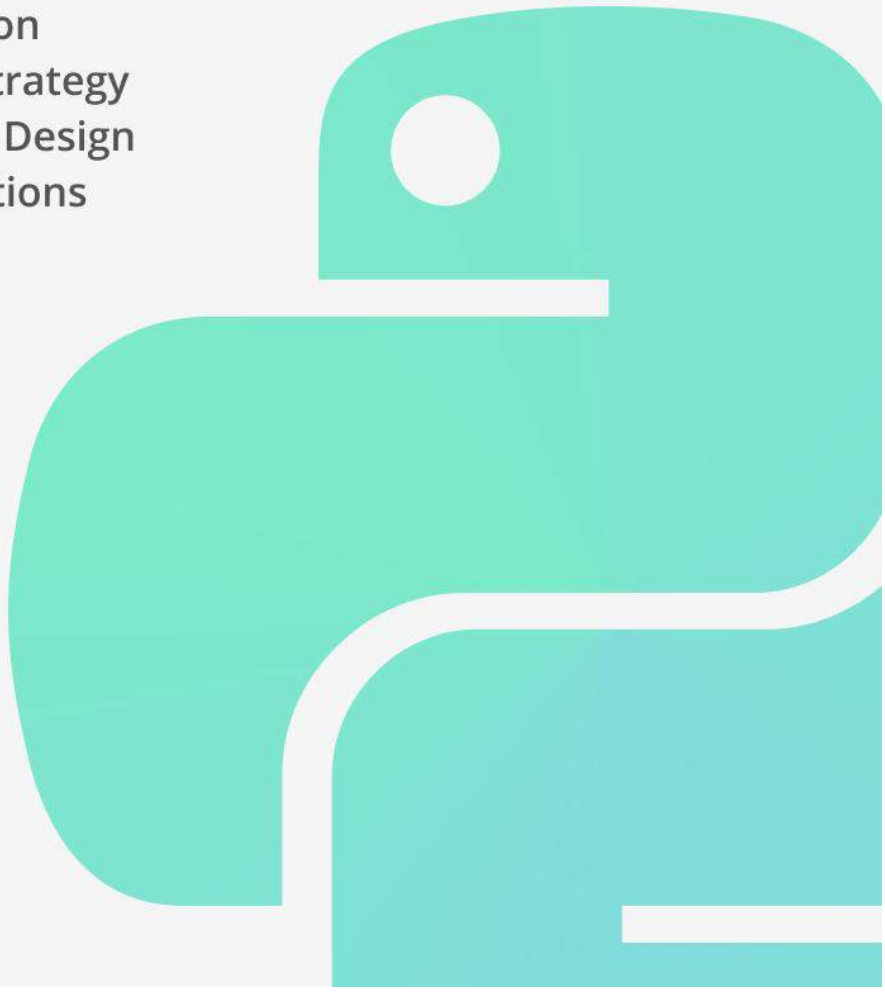
CS 4 AI

A 100% Free and Complete Reference Book

CBSE Class 12 2021-22 Term 1 Edition

Arjun Singh Sodhi & Sammarth Kumar

- Detailed Notes and Revision
- Detailed SQP Analysis & Strategy
- Interactive and Appealing Design
- MCQs & Case Based Questions



A Q-Programming Publication

Introduction

Q-Programming is dedicated to providing numerous resources for secondary school students. This free book is a complete guide for the CBSE 2021-22 Term 1 Board Exam for Computer Science with Python.

Make sure to check out our YouTube channel for detailed lectures on each topic: https://youtube.com/qprogramming.

Visit our website for many more free resources for the CBSE curriculum: https://qprogramming.net.

Book Structure

The book has been divided into two sections: **notes** and **questions**.

The notes section contains all the relevant notes from NCERT and the official syllabus, and additional information as well for your knowledge (which could potentially appear in the exam). Examples of code have been given with syntax highlighting to demonstrate concepts. You are encouraged to attempt the examples on your system and also play around with the code for better understanding. We have also consulted the computer science reference books authored by Sumita Arora (Computer Science with Python) and Preeti Arora (Computer Science with Python).

The questions section consists of chapter wise Multiple Choice Questions (MCQs) and Case Based Questions (CBQs). The MCQs include information from every topic, subtopic and relevant information that can appear on the test and will cover almost all of the information that could be tested. The CBQs have been created according to the pattern of questions released by CBSE.

The authors would like to thank the Q-Programming team and their friends, family and teachers for their support in this endeavour.

User Agreement

This publication is the property of the authors and may not be circulated or sold for profit without the authors' consent and cannot be reproduced or copied without the written permission from **both** authors. Any violation of this condition is liable for legal action.

A large effort has gone into making this publication and we have tried to ensure that all information is accurate and that almost all the relevant material has been covered. In the event that you feel certain material has been left out or is incorrect, you may email us at contact@qprogramming.net for such queries.

It is notified that by using this publication, you agree to not hold Q-Programming or the authors accountable for any form of errors or discrepancies and agree to all the conditions listed above.

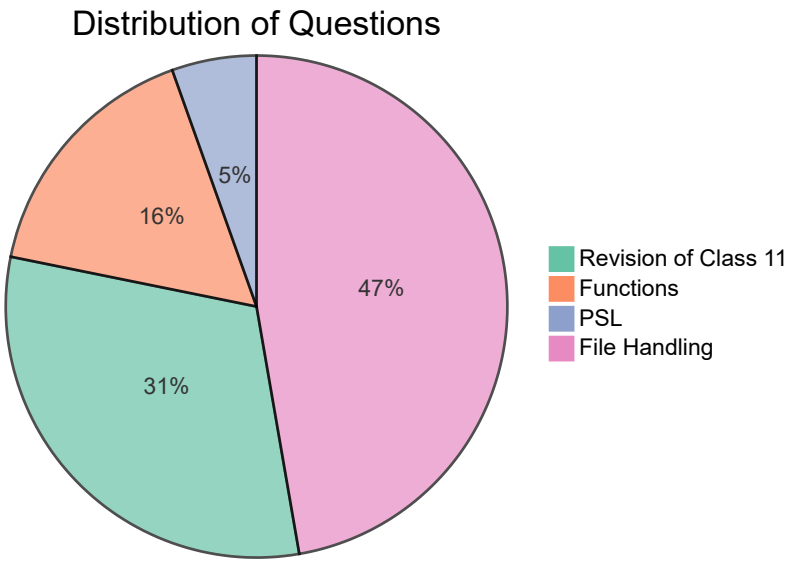
Table of Contents

Sample Paper Analysis

We analysed the CBSE 2021 Term 1 Sample Paper to see the distribution of questions and the different types of questions that are present. We have created a table of the distribution of questions according to the chapters in this book. Note that since some questions involve concepts from multiple chapters, there may be a little variation in the distribution, depending on the opinion of different people. Also, the sample paper may not be identical to the actual question paper in terms of weightage of different topics since no official weightages have been allotted to each individual topic. We do not claim this. We therefore state that this analysis strictly applies to the **sample paper** only and may or may not hold true for the actual CBSE Term 1 paper.

Chapter Wise Distribution

	Revision of Class 11	Functions	Python Standard Library Functions	File Handling
Section A	1,2,3,9,10,22,23,25	15,16,17	8	4,5,6,7,11,12,13,14,18,19,20,21,24
Section B	26,28,29,30,35,41,49	33,34,36,37,39,44	31,38	27,32,40,42,43,45,46,47,48
Section C	53, 54			50,51,52,55
Total	17	9	3	26



As can be seen, **almost 50% of the paper** is from file handling which makes it a **very** important unit. The questions in file handling are of multiple kinds ranging from factual questions, syntax questions, output questions to case based questions.

To tackle these questions, knowledge of the syntax of each function is required, especially default values of parameters. Many of the questions give the content of the file, some code and then ask for the output.

In the revision of class 11 portion, questions have come from different areas and it is important to know everything. Operator precedence questions, slicing and indexing, loops, conditionals, identifier naming are all important concepts. Small facts are also important, which are available in our notes.

From the functions unit, questions are based on syntax of function and output questions. Output questions involve local and global variables, mutable data types, etc.

From Python Standard Library functions, a random module question appeared and also simple questions such as data type returned by input and the use of list methods. While the weightage of this chapter is quite low, it is a good idea to have knowledge of the different topics within it. We have included a lot of extra information in this chapter for such reasons.

Difficulty

The paper contains some very simple questions which should be solved in seconds, such as importing csv module, name the extension of Python files, etc.

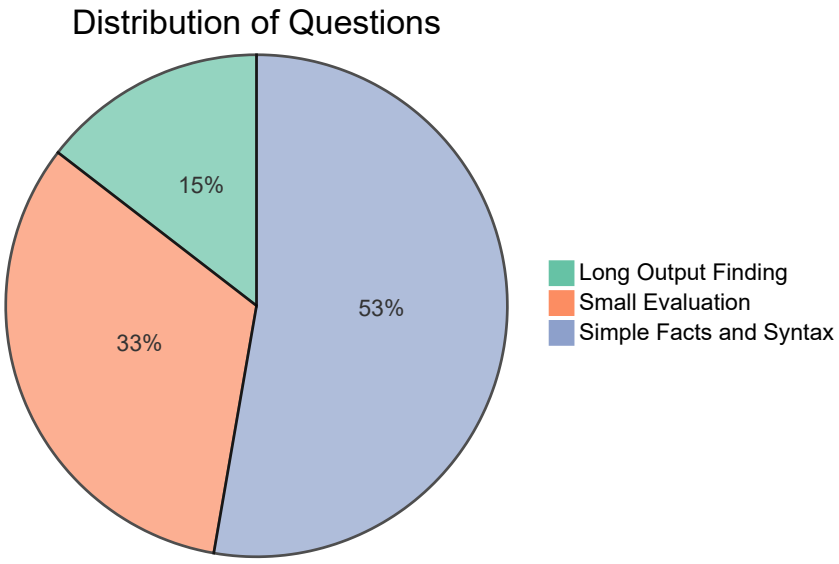
The lengthier questions are the evaluation and output finding questions, mostly located in Section B. These questions will definitely take more time than the others and should be done carefully.

Strategy

In our opinion, the best way to attempt a paper like this is to first solve the simple questions which can be done quickly, such as simple factual questions. Secondly, the case study consists of short questions so it should also be done in the beginning. After this, the slightly harder questions such as indexing or expression evaluation can be solved. In the end, the longer output finding questions can be attempted with sufficient breathing room. Since there is a choice available in questions, it is advisable to go for simple questions over the longer output finding questions, unless you are unsure of the simpler ones.

The following is the breakdown of the questions in the aforementioned categories:

Section	Simple Facts and Syntax	Small Evaluation Questions	Long Output Finding Questions
A	1,2,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,20,21,22,23,25	3,10,24	
B	48	26,27,28,29,30,31,32,33,34,35,40,42,43,47,49	36,37,38,39,41,44,45,46
C	50,51,52,53,54,55		
Total	29	18	8



From this table it is clear that Section A and Section C can be done fast while Section B will take longer. Remember, in Section B you have to attempt 20/24 questions so it is possible to avoid at most 4 long output questions.

If you don't think this strategy works for you, don't worry - everyone can have their own preferred methods - this is just a suggestion.

Chapter 1: Revision of Class 11

Tokens

There are 5 kinds of tokens - individual units in a program, also known as a **lexical units**.

1. Keywords

These are words with a predefined purpose. There are 33 **official** keywords.

2. Identifiers

These are names for different objects. There are certain **rules** which must be followed while creating an identifier:

1. They *cannot* be a keyword
2. They can only have letters, numbers and underscores
3. They *cannot* begin with a number, but may contain numbers
4. They cannot contain special characters

Examples of acceptable identifiers: file, some_variable, list1

Examples of unacceptable identifiers: 1var, var-1, if

3. Literals

They are values that have a fixed value. They can be classified as String, Numeric, Boolean, None literals.

4. Operators

They are used to perform an operation on operands(variables, literals, etc.)

5. Punctuators

They are symbols used to organize statements. Example: ':', '=', "@", "( )", "[ ]"

Variables

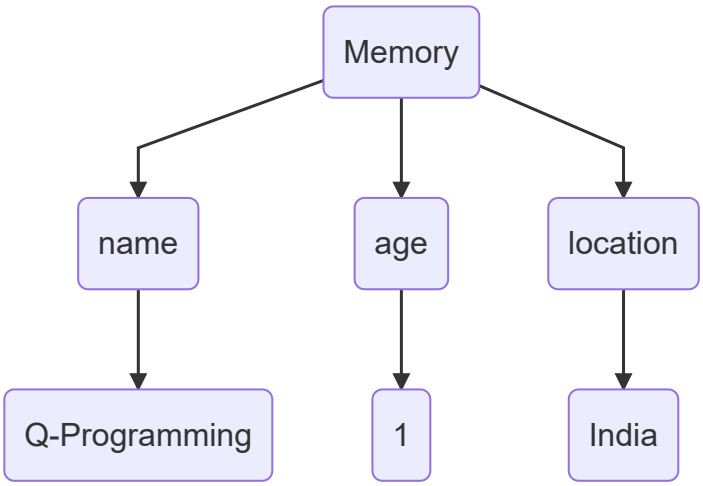
Variables store different values and can be of different data types.

A variable can be **defined** as follows: `var = 123`

Here, the **identifier** is `var`, the **punctuator** to assign the **literal** to the variable is `=` and the **literal** is `123`.

The identifier is the name of the variable and will point to the location of the object in the memory.

Example



If we want to assign values to different variables at once we can do it for:

1. **Multiple variables with the same value**

```
>>> a = b = c = d = 100
```

2. **Multiple variables with different values**

```
>>> a,b,c,d = 1,2,3,4
```

   Here, a,b,c,d are respectively assigned the values of 1,2,3,4.

Swapping

In Python we can swap the values of variables as follows:

```
>>> a = 10
>>> b = 20
>>> a,b = b,a
>>> print(a,b)
20 10
```

Dynamic Typing

In Python, the data type of a variable is not fixed and can be changed. For example, we can do

```
x = 2
x = "hello"
```

and it will be acceptable. This is in contrast to **static typing** which places constraints on data types of variables.

Input

We use the `input()` function to take in input which returns the input as a **string**.

Example

```
>>> var = input("Enter name: ")
Enter name: qprogramming
```

Now, `var` is assigned a value of `"qprogramming"`.

If we want to change the data type we can use **type casting functions** (to be covered later in this chapter).

Output

We use the `print()` function to display output.

Example

```
>>> print("hello")
hello
```

There are certain **arguments** we can pass to the `print()` function for different purposes.

`sep`

The `sep` argument is used to separate multiple arguments in the `print()` function by a **string**. By default, the value of `sep` is `" "` which is a whitespace character.

We can change the value of `sep`.

Example

```
>>> print(1,2,3,sep="##")
1##2##3
```

`end`

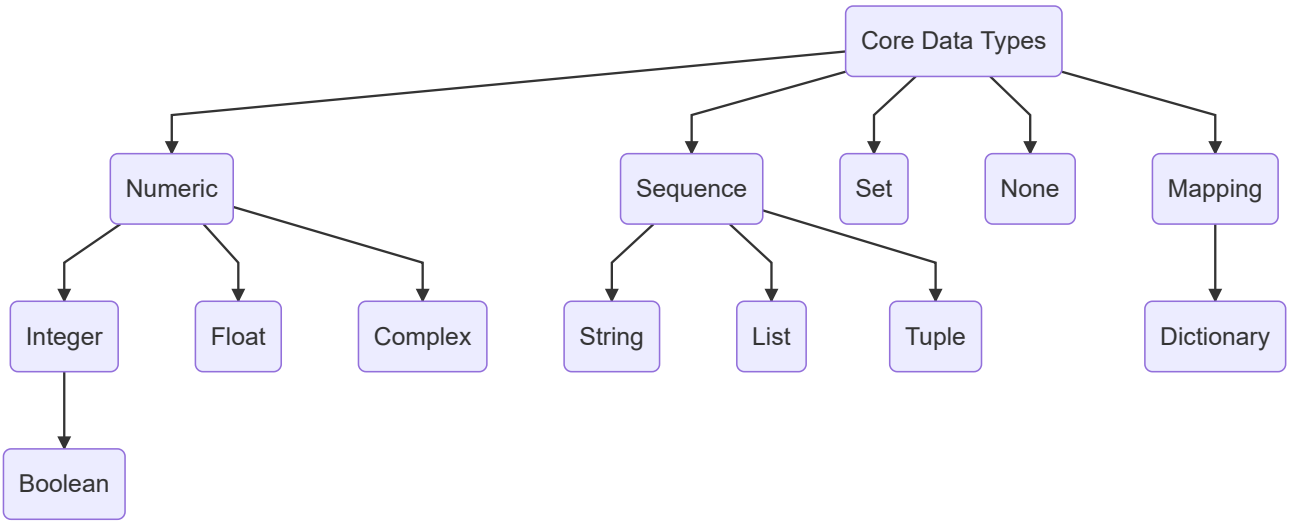
The `end` argument is used to add a string to the end of the printed output. By default it is `"\n"` (newline).

Example

```
>>> print(123,end="hello")
123hello
```

Fun Fact: You can add diffferent data types in the print statement as separate arguments.

Data Types



Numeric

Integers

Integers are the same as integers in mathematics: positive and negative whole numbers.

Integers may also be represented in binary, octal and hexadecimal form:

Decimal	Binary	Octal	Hex
17	0b10001	0o21	0x11

Boolean

Boolean can be considered a form of integers.

True is equivalent to 1 and **False** is equivalent to 0.

It is to be noted that empty sequences are also equivalent to **False** while sequences with elements are equivalent to **True**.

Floats

Floating point numbers resemble fractions represented in decimal form and include a decimal point.

Complex

Complex numbers are written in the form `a + bj` where `a` is the real component and `bj` is the imaginary component. Note that we use a `j` rather than the mathematical notation of `i`.

Sequence

Sequences are combinations of literals.

Strings

Strings are ordered and immutable sequences of characters.

Lists

Lists are ordered and mutable sequences of objects.

Tuples

Tuples are ordered and immutable sequences of objects.

Set

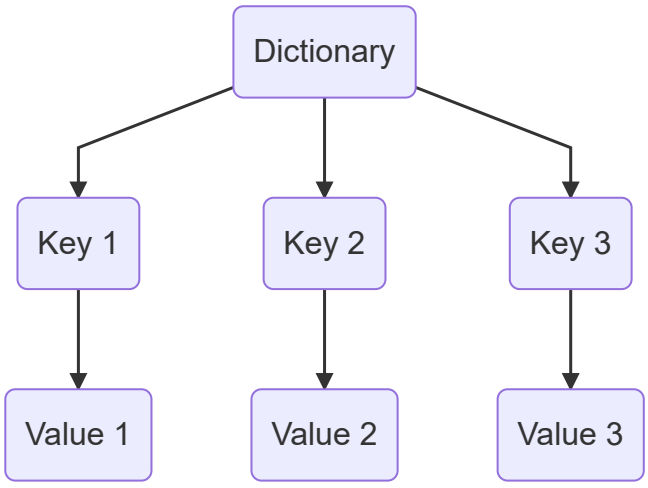
Sets are unordered and mutable collections of values wherein no repetitions of values occur.

None

`None` is a special data type unique to Python which represents a null value.

Mapping

Mapping data type consists of mapping keys to values, thus creating **key-value** pairs. In Python, we have the Dictionary data type for mapping which are ordered and mutable sequences of key-value pairs.



`type()`

The `type()` function can be used to find the data type of an object.

Example

```
>>> type(1)
<class 'int'>
>>> type("hello")
<class 'str'>
```

Mutability

Mutability refers to the ability to change the value of an object without changing its location in the memory.

Mutable	Immutable
Can change values without changing location	Cannot change values without changing location
Eg: list, dictionary	Eg: string, int, tuple

We use the `id()` function to check the memory address of an object. Literals have fixed addresses in the memory.

Expressions

Expressions are valid combinations of operators, literals and variables.

For example, `a + b * 100 - c` is considered as an expression. Here, a,b,c are variables, 100 is a literal and +, *, - are operators.

Operators

1. **Arithmetic**

Operator	Operation	Return Type
+, -, *	addition, subtraction, multiplication	If both are int, `int` If one or both float, `float`
/	division	`float`
//	floor division (rounds down to integer)	if both int, `int` If one or both float, `float`
%	modulus (remainder)	If both int, `int` If one or both float, `float`
**	Exponentiation	If both int, `int` If one or both float, `float`

2. **Comparison**

Operator	Comparison	Return Type
<, >	less/greater than	`bool`
<=, >=	less/greater than equal to	`bool`
==	equal to	`bool`
!=	not equal to	`bool`

3. **Logical**

Operator	Operation	Return Type
not	reverses boolean value	`bool`
and	checks if both are true	`bool`
or	checks if one or more are true	`bool`

4. **Identity**

Operator	Checks If	Return Type
is	objects are same	`bool`
is not	objects are not the same	`bool`

5. **Membership**

Operator	Checks If	Return Type
in	object in sequence	`bool`
not in	object is not in sequence	`bool`

Operator Precedence

The following table contains the precedence of operators while evaluating expressions:

Operator	Use
()	grouping
**	Exponent
~x	Bitwise NOT
+x, -x	Unary operation
*,/,//,%	Multiplication, division, floor division, modulus
+, -	Addition and subtraction
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
<, <=,>,>=,!=,==,is,is not	Relational Operators and Identity Operators
not	Boolean NOT
and	Boolean AND
or	Boolean OR

When we evaluate an expression, the operator precedence comes into play and we evaluate operations of higher precedence first. It is to be noted that when the same operator occurs successively, the expression is evaluated from left to right **except** for exponents.

So,

```
>>> 1/2/3
0.16666666666666666
```

as its is equal to `(1/2)/3` = `0.5/6` but,

```
>>> 2**3**2
512
```

because it is actually `2**(3**2)` = `2**9`.

Type Casting

This is also known as explicit type conversion and is used to convert an expression to a specific data type.

For example if we want to convert `105` to a string, we can us the `str()` function:

```
x = 105
x = str(x)
```

Now, `x` is a string and equal to `'105'`.

The following type casting functions can be used:

Function	Purpose
`int()`	To convert to integer
`float()`	To convert to float
`str()`	To convert to string
`bool()`	To convert to boolean
`list()`	To convert to list
`tuple()`	To convert to tuple
`dict()`	To convert to dictionary

Comments

Comments are lines in a program that are not executed.

Comments are useful for the following:

1. To add meaningful documentation to a code
2. To "comment out" code so that it remains in the file but is not executed

Single Line

We use a `#` to denote a single line comment.

Example

```
# this is a single comment
print(1) # content after the hash will not be executed
```

Output

```
1
```

Multiline

We use a pair of 3 single or 3 double quotes for multiline comments.

Example

```
"""this is a multiline
comment"""
print("hello")
```

Output

```
hello
```

Flow of Control

Flow of control is used to control the way a program is executed based on certain conditions and parameters.

We use conditionals and loops for these purposes. These are **blocks** of code which have certain conditions associated with them that decide how those blocks of code are to be executed.

A block of code consists of code **indented** in a **compound statement header**. What is a compound statement? It is a group of statements that are executed together.

```
<coumpound header>:
    <statements>
```

The statements within the compound statement are indented by default to **one tab space**.

Conditionals

Conditionals consist of `if`, `elif` and `else` statements

`if`

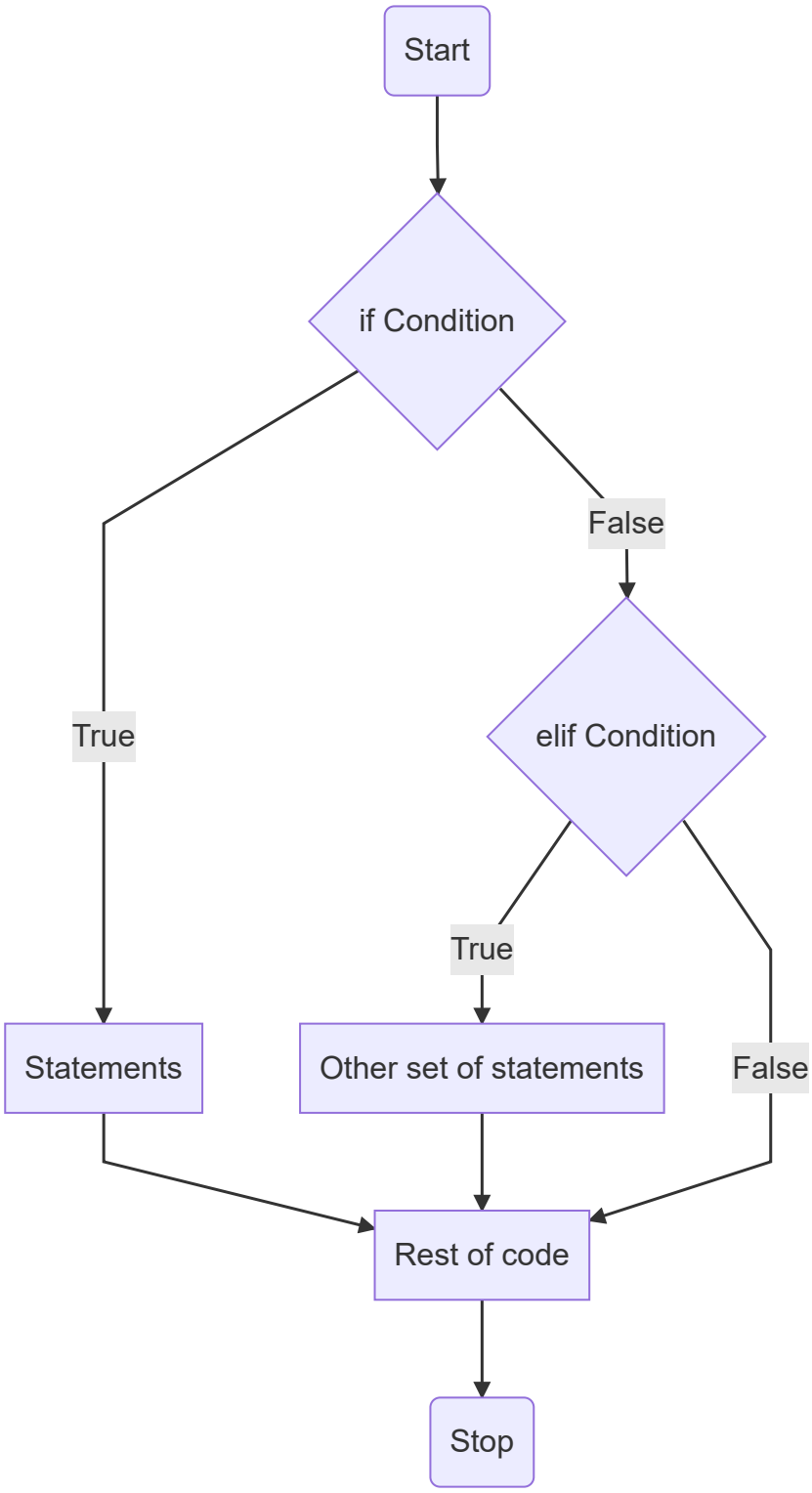
This is the first condition checked in a conditional construct. If the condition is met, the code within it is executed.

`elif`

This is an alternate and optional condition provided if the previous condition is not satisfied. We can use as many `elif` statements as we please.

`else`

This is the code that's executed if none of the conditions are satisfied.



The syntax is:

```
if <condition 1>:
    <statements>
elif <condition 2>:
    <statements>
elif <condition 3>:
    <statements>
else:
    <statements>
```

Note that the `else` and `elif` statements may or may not be used, but for a condition to be checked, the `if` is required.

Nested `if` statements

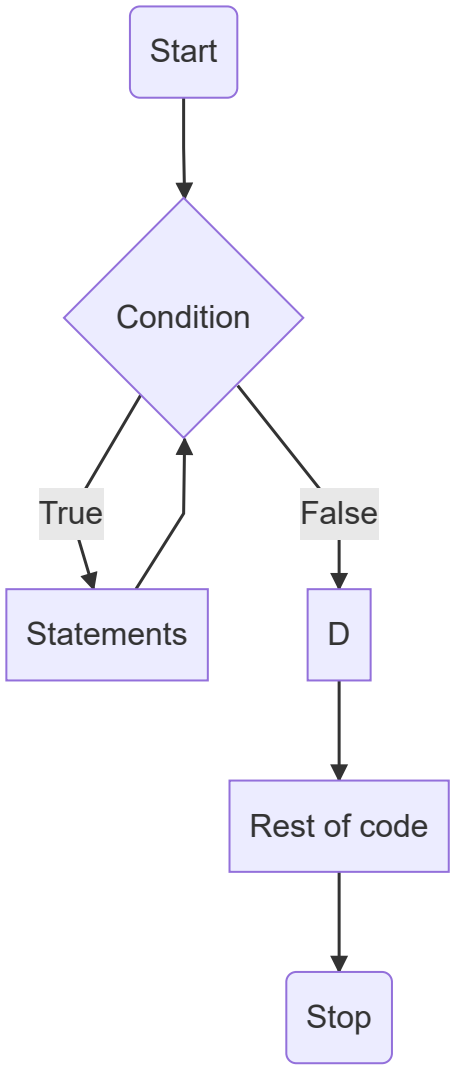
We can have `if` statements nested within `if` statements.

```
if <condition 1>:
    if <condition 2>:
        <statements>
```

This is simple logic - if the first condition is satisfied, the second condition is checked and then the statements are accordingly executed or not.

Loops

Loops are used to repeat blocks of code as long a condition is satisfied.



`for` loop

The syntax is

```
for <control variable> in <sequence>:
    <statements>
```

For each iteration, the **control variable** assumes the respective element in the **sequence**. For example, in the 10th iteration of a loop, the control value will assume a value equal to the 10th element in the sequence. This is known as iterating over a sequence.

We can also use the `range()` function. It has the syntax `range(<start>,<stop>,<step>)`.

The start value is the starting value of the range (by default 0) and the stop value is one value after the last value in the range. The step value specifies a number n which gives every nth element in the range (by default 1).

So, `range(0,5)` will give us 0,1,2,3,4 and `range(0,5,2)` will give us 0,2,4. Note that the range function returns an iterable which we can use in a loop.

Example

```
for i in range(0,5,2):
    print(i)
```

Output

```
0
2
4
```

`while` loop

The syntax is

```
while <condition>:
    <statements>
```

As long as the condition is satisfied, the statements will be executed in each iteration.

Jump Statements

Jump statements are used to control the way loops execute.

`break`

The `break` statement is used to exit a loop completely.

Example

```
x = 0
while True:
    x += 1
    if x == 4:
        break
    print(x)
```

Output

```
0
1
2
3
```

When `x` is assigned a value of `4`, the condition is satisfied and the break is executed, thus terminating the loop and not printing `4`.

`continue`

The `continue` statement is used to skip a particular iteration of a loop.

Example

```
for i in range(1,6):
    if i == 3:
        continue
    print(i)
```

Output

```
1
2
4
5
6
```

Here, when the control variable `i` is assigned a value of 3, the program skips to the next iteration when the `continue` statement is executed, and thus `3` is not printed.

Nested Loops

Nested loops are simply loops within loops.

Example

```
for i in range(5):
    for j in range(5):
        print("*", end="")
    print()
```

Output

```
*****
*****
*****
*****
*****
```

In this case, the second loop runs 5 times and the code within it runs 5 times for each execution of the second loop, and thus the statements are executed 25 times. This gives us the result that number of times code within `n nested loops of m iterations is executed is equal to m^n.

Strings

1. **Single line:** Enclosed in pairs of single or double quotes.

2. **Multiline:** Enclosed in pairs of three single or double qu

Indexing and Slicing

Each element of a string is assigned a unique **index** which is simply a number indicating the position of that element in the string sequence.

The value of the index is different starting from the left or right side of the string:

	s	t	r	i	n	g
From Left	0	1	2	3	4	5
From Right	-6	-5	-4	-3	-2	-1

Note that from the right indices start from -1 but from the left they start from 0.

To access a particular character, we can use its index position.

Example

```
>>> string = "string"
>>> string[0]
's'
>>> string[-2]
'n'
```

Slicing is an operation to extract certain portions of a string.

We use square brackets again to do so. The syntax is `string[<start>:<stop>:<step>]` similar to the `range()` function's syntax.

Let's first see how we can use slicing for positive values of `start` and `stop`.

Example

```
>>> string = "qprogramming"
>>> string[0:2]
'qp'
```

Notice how it prints only the value at the 0th and 1st index.

What if we use negative indices?

Example

```
>>> string = "qprogramming"
>>> string[1:-1]
'programmin'
```

Here, the program "slices" the values from the first index from the left to the first index from the right.

Let's see how we can use the step value.

Example

```
>>> string = "qprogramming"
>>> string[0:6:2]
'qrg'
>>> string[6:3:-1]
'arg'
```

As you can see, it will output every second character in the range of the 0th and 6th index.

In the second example, we got the 6th 5th and 4th index in reverse order. Why? Because it prints the 6th character then the (6-1)th character and finally the (6-2)th character as the step value is negative 1.

When we don't give the string a start or stop value, the default start is 0, the stop is the length of the list and the step is 1.

Example

```
>>> string = "qprogramming"
>>> string[:]
'qprogramming'
>>> string[:5]
'qprog'
>>> string[5:]
'ramming'
```

Traversing

A string is a sequence of characters and when we iterate over a string with a for loop, the control variable will be assigned one character in each iteration.

Example

```
string = "hello"
for i in string:
    print(i)
```

Output

```
h
e
l
l
o
```

Operations

Concatenation

We can use the `+` operator to concatenate (join) strings:

```
>>> "hello" + " world"
'hello world'
```

Note that the whitespace is counted as a character.

We cannot concatenate strings with other data types as it will produce an error

```
>>> "s" + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Replication

We can use the `*` to replicate content in a string by keeping a string on the left and an integer on the right. If the integer is less than or equal to zero, a blank string will be returned.

```
>>> "hi" * 5
'hihihihihi'
```

Membership

We can use the `in` operator to check if a substring is in a string:

```
>>> "h" in "hello"
True
>>> "h" not in "hello"
False
>>> "i" in "hello"
False
```

Lists

Indexing and Slicing

These operations are similar to those in strings. Each element in a list is assigned a unique index in the same way as in strings.

Example

```
>>> lst = [1,2,3,4,5]
>>> lst[2]
3
>>> lst[-4]
2
>>> lst[:]
[1, 2, 3, 4, 5]
>>> lst[2:6]
[3, 4, 5]
>>> lst[6:2:-1]
[5, 4]
```

Since list is a mutable data type, we can change the value of an element as follows:

Example

```
>>> lst = [1,2,3]
>>> lst[2] = 100
>>> print(lst)
[1, 2, 100]
```

Nested Lists

Nested lists are lists within lists for example `[[1,2],[3,4]]`

To access values in nested lists, we need to first use the index of the nested list in the parent list and then index the value in the nested list.

Example

```
>>> lst = [[1,2],[3,4]]
>>> lst[0][1] # second element in first nested list
2
```

Traversing

We can use a `for` loop to traverse a nested list. The control variable is assigned the value of the element in each respective iteration.

Example

```
lst = [1,2,3,4]
for i in lst:
    print(i)
```

Output

```
1
2
3
4
```

Operations

Concatenation

This is again similar to concatenation in strings. We can concatenate lists with lists **only**

Example

```
>>> lst1 = [1,2,3]
>>> lst2 = [4,5,6]
>>> lst1 + lst2 + [7,8]
[1, 2, 3, 4, 5, 6, 7, 8]
```

Replication

Example

```
>>> lst = [1,2,3]
>>> lst * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Membership

Example

```
>>> lst = [1,2,"3", True]
>>> 1 in lst
True
>>> True in lst
True
>>> 2 not in lst
False
>>> [1,2] in lst
False
```

Notice that we cannot check for a group of specific elements, unlike in strings.

Deletion

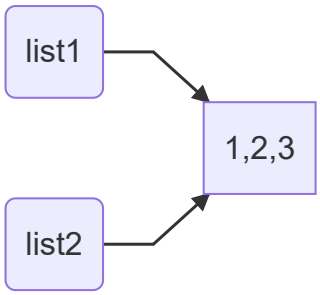
We can delete elements or slices using the `del` keyword.

Example

```
>>> lst = [1,2,3,4,5]
>>> del lst[0]
>>> lst
[2, 3, 4, 5]
>>> del lst[1:]
>>> lst
[2]
```

List Copies

We can duplicate the entries of one list `list1 = [1,2,3] to a second by simply` list2 = list1`, however, both identifiers point to the same object in the memory which means if we change the value of` list1 `the value of` list2 ` will also change. This is known as a **shallow copy**. To make a **true** copy we need to make a **deep copy** which results in a new memory location. There are three methods we can use to make a deep copy.



Method 1: `list()`

```
>>> lst = [1,2,3]
>>> id(lst)
3040792062144
>>> lst2 = list(lst)
>>> id(lst2)
3040792063680
```

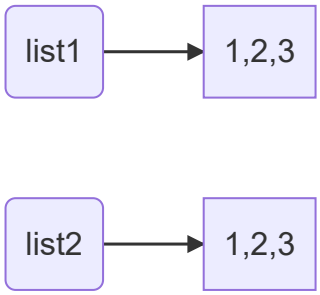
Method 2: slicing

```
>>> lst = [1,2,3]
>>> id(lst)
3040792008384
>>> lst2 = lst[:]
>>> id(lst2)
3040792062144
```

Method 3: `copy.copy()`

```
>>> import copy
>>> lst = [1,2,3]
>>> id(lst)
3040792062464
>>> lst2 = copy.copy(lst)
>>> id(lst2)
3040792008384
```

This is what a copy would look look diagrammatically:



Tuples

Tuples are immutable sequences.

A key point to know is how to create single element tuples.

The statement `a = (1)` will not create a tuple, it will assign an `int` 1 to `a`.

To assign a single element we use `a=(1,)`.

An interesting thing to note is that `a = 1,2,3` assigns a tuple to `a`.

If we want to create an empty tuple we use `a = tuple()` or `a = ()`

Indexing and Slicing

Tuple indexing and slicing is similar to that of lists, except we cannot change values as tuples are immutable

Traversing

Traversing a tuple with a `for` loop is similar to that of lists.

Operations

Tuple operations are the same as in lists

Unpacking Tuples

Tuple unpacking is simply creating individual values from the elements of a tuple.

Example

```
>>> a,b,c = (1,2,3)
>>> print(a,b,c)
1 2 3
```

The number of identifiers on the left hand side should be equal to the number of elements in the tuple, otherwise we get an error.

Example

```
>>> a,b = (1,2,3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
```

Tuples v/s Lists

	Tuple	List
Syntax	Round brackets	Square brackets
Mutability	Immutable	Mutable

Comparison in Sequences

Comparison operators work with sequences. They compare the respective values of elements in sequences.

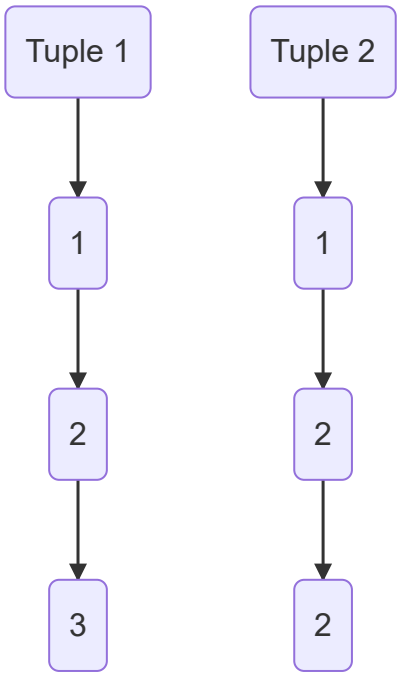
With numerical values

When using comparison operators with sequences of numerical values such as tuples and lists, each respective lement is compared based on the value of the number.

Example

```
>>> (1,2,3) > (1,2,2)
True
```

Here, the first 2 respective elements are equal, but the third element of the first tuple is greater than the third element of the second tuple.



With String Values

The relational operators compare strings with normal comparison operators. However, in the case of a string, it will compare the **ASCII** value of each character from left to right.

Example

```
>>> "a" > "b"
False
```

`"a"` has an ASCII value of 97 while `"b"` has a value of 98, so the value of "b" is greater.

Example

```
>>> "abc" > "aac"
True
```

In this case, the first characters in both strings have the same ASCII value, so Python checks the second characters. Since the ASCII value of `"b"` is greater, the output will be `True`. Therefore, Python will keep checking the values until the comparison operator can be used.

If we have a list containing strings then it will compare each element of the list respectively.

Example

```
>>> ["a", "b", "c"] > ["A", "b", "c"]
True
>>> ["A", "b", "C"] > ["A", "b", "c"]
False
```

Note that we **cannot** try to compare a numerical value as it will produce a `TypeError`

Dictionaries

Accessing Elements

We know that dictionaries consist of **key-value pairs**. To access a value we need to use the key. This is similar to indexing, except instead of an index we use a key to get the value associated with that key.

Example

```
dict1 = {"name": "qprogramming", "location": "India"}
print(dict1["location"])
```

Output

```
India
```

Since dictionaries are mutable, we can change the values associated with different keys.

Example

```
dict1 = {"name": "qprogramming", "location": "India"}
dict1['name'] = 'Q-Programming'
print(dict1)
```

Output

```
{'name': 'Q-Programming', 'location': 'India'}
```

Note that the key can be **any immutable data type**, and that includes integers, floats, tuples, strings, etc., but each key must be unique.

We can also add pairs to a dictionary as follows.

Example

```
dict1 = {"name": "Q-Programming", "location": "India"}
dict1['site'] = 'qprogramming.net'
print(dict1)
```

Output

```
{'name': 'Q-Programming', 'location': 'India', 'site': 'qprogramming.net'}
```

Traversing

If we use a for loop to iterate over a dictionary, it will iterate over its keys.

Example

```
dict1 = {"name": "qprogramming", "location": "India"}

for i in dict1:
    print(i)
```

Output

```
name
location
```

To traverse the key-value pairs of a dictionary, we take the help of the `.items()` method which returns an iterable containing key value pairs as tuples:

Example

```
dict1 = {"name": "qprogramming", "location": "India"}

for key, value in dict1.items():
    print(key, value)
```

Output

```
name qprogramming
location India
```

Operations

Membership

Membership operators allow us to check whether a key exists in a dictionary.

Example

```
>>> dict1 = {"name": "qprogramming", "location": "India"}
>>> "name" in dict1
True
>>> "age" in dict1
False
```

Deletion

We can use `del` to delete a key-value pair.

Example

```
>>> dict1 = {"name": "qprogramming", "location": "India"}
>>> del dict1["name"]
>>> dict1
{'location': 'India'}
```

We can also use the `pop()` method to delete a key-value pair.

Example

```
>>> dict1 = {"name": "qprogramming", "location": "India"}
>>> dict1.pop("location")
'India'
>>> dict1
{'name': 'qprogramming'}
```

Notice that it returns the value

Errors

Sometimes an error may occur in the program and can be classified into three types:

1. **Syntax Errors**

   These occur when the user has not typed the correct syntax as per the Python conventions.

   Example

   ```
   >>> print2)
     File "<stdin>", line 1
       print2)
             ^
   SyntaxError: unmatched ')'
   ```

2. **Logical Errors**

   These occur when the desired output is not given, even if the code is running.

   Example

   I want to get the output as `hello`

   ```
   >>> var = "hello"
   >>> var2 = "bye"
   >>> print(var)
   bye
   ```

Here I used the incorrect variable name, so got the wrong output even though the program ran properly.

3. **Runtime Errors**

   These errors occur when the syntax may be correct but there is a problem in the execution. Common runtime errors are `ZeroDivisionError` when we try to divide by zero, `NameError` when we try to use a variable that has not been defined or `TypeError` when we try to perform operations on different data types such as concatenation or when we try to perform an operation not supported by a data type, such as item assignment in strings.

   Example

```
>>> print(var3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'var3' is not defined
```

Solved Examples

1. Find the invalid identifier from the following

(SQP 2021)

a) none
b) address
c) Name
d) pass

Ans: d) pass

Pass is a keyword and cannot be used as an identifier.

2.Consider a declaration `L = (1, 'Python', '3.14')`. Which of the following represents the data type of L?

(SQP 2021)

a) list
b) tuple
c) dictionary
d) string

Ans: b) tuple

Tuples are ordered sequences of objects enclosed within round brackets.

3. Given a Tuple `tup1= (10, 20, 30, 40, 50, 60, 70, 80, 90)`.

What will be the output of `print(tup1[3:7:2])`?

(SQP 2021)

a) (40,50,60,70,80)
b) (40,50,60,70)
c) [40,60]
d) (40,60)

Ans: d) (40,60)

It will return a tuple containing the 3rd and 5th index values since there is a step value of 2.

4. Which of the following operator cannot be used with string data type?

(SQP 2021)

a) +
b) in
c) *
d) /

Ans: d) /

All operators can be used with strings except `/`.

5. Consider a tuple `tup1 = (10, 15, 25, and 30)`. Identify the statement that will result in an error.

(SQP 2021)

a) print(tup1[2])
b) tup1[2] = 20
c) print(min(tup1))
d) print(len(tup1))

Ans: b) tup1[2] = 20

Tuples are immutable so we cannot change the value of an element using the index.

6. Which one of the following is the default extension of a Python file?

(SQP 2021)

a) .exe
b) .p++
c) .py
d) .p

Ans: c) .py

`.py` file extensions are used for modules.

7. Which of the following symbol is used in Python for single line comment?

(SQP 2021)

a) /
b) /*
c) //
d) #

Ans: d) #

`#` is used for single line comment.

8. Which of these about a dictionary is false?

(SQP 2021)

a) The values of a dictionary can be accessed using keys
b) The keys of a dictionary can be accessed using values
c) Dictionaries aren't ordered
d) Dictionaries are mutable

Ans: b) The keys of a dictionary can be accessed using values

Dictionaries are mutable and ordered sequences of key-value pairs. Individual values can be accessed using their respective keys.

9. What is the output of following code:

```
T=(100)
print(T*2)
```

(SQP 2021)

a) Syntax error
b) (200,)
c) 200
d) (100,100)

Ans: c) 200

`T = (100)` assigns an `int` 100 to `T`.

10. Identify the output of the following Python statements.

```
x = [[10.0, 11.0, 12.0],[13.0, 14.0, 15.0]]
y = x[1][2]
print(y)
```

(SQP 2021)

a) 12.0
b) 13.0
c) 14.0
d) 15.0

Ans: d) 15.0

`x[1][2]` accesses the 3rd element of the second list which is `15.0` i.e. the 2nd index in the list with index position 1 in the parent list.

11. Identify the output of the following Python statements.

```
x = 2
while x < 9:
    print(x, end='')
    x = x + 1
```

(SQP 2021)

a) 12345678
b) 123456789
c) 2345678
d) 23456789

Ans: c) 2345678

As long as `x < 9` the program will print a number.

It starts from `2` and adds one in each iteration. When `x` is `8`, it prints `8` but will terminate afterwards since `8+1` = `9` and `9 < 9` will be `False`

12. Identify the output of the following Python statements.

```
b = 1
for a in range(1, 10, 2):
    b += a + 2
print(b)
```

(SQP 2021)

a) 31
b) 33
c) 36
d) 39

Ans: c) 36

Let's break this down. The program adds the value of `a + 2` to `b` in every iteration.

We know that `range(1, 10, 2)` will produce an iterable containing `1, 3, 5, 7, 9` which means there will be 5 iterations.

Let us assume the final value is x .

Then $x = 1 + (1 + 2) + (3 + 2) + \ldots + (9 + 2)$.

We can rewrite this as $x = 1 + (1 + 3 + 5 + 7 + 9) + 5 \times 2 = 1 + 25 + 10 = 36$.

13. Evaluate the following expression

```
16 - (4 + 2) * 5 + 2**3 * 4
```

(SQP 2021)

a) 54
b) 46
c) 18
d) 32

Ans: c) 18

Let's evaluate the expression according to operator precedence.

Step 1: Evaluate `(4+2)`

`16 - 6 * 5 + 2**3 * 4`

Step 2: Evaluate `2**3`

`16 - 6 * 5 + 8 * 4`

Step 3: Evaluate `6 * 5` **and** `8 * 4`

`16 - 30 + 32`

Final Ans: `18`

14. Find the output of the following code:

```
Name="PythoN3.1"
R=""
for x in range(len(Name)):
    if Name[x].isupper():
        R=R+Name[x].lower()
    elif Name[x].islower():
        R=R+Name[x].upper()
    elif Name[x].isdigit():
        R=R+Name[x-1]
    else:
        R=R+"#"
print(R)
```

(SQP 2021)

a) pYTHOn##.
b) pYTHOnN#.
c) pYTHOn#.
d) pYTHOnN.#

Note: The original question contains an "@" instead of a "." which we believe is a typo and have accordingly adjusted the answer choices.

Ans: b) pYTHOnN#.

This is a long question. Let's do it step by step for each iteration.

Iteration	Value of `Name[x]`	Explanation	String
1	P	P is uppercase so a lowercase p is added	p
2	y	y is lowercase so an uppercase Y is added	pY
3	t	t is lowercase	pYT
4	h	h is lowercase	pYTH
5	o	o is lowercase	pYTHO
6	N	N is uppercase	pYTHOn
7	3	3 is a digit so N is added since it is the character before	pYTHOnN
8	.	. does not satisfy any condition so # is added	pyTHOnN#
9	1	1 is a digit so . is added	pYTHOnN#.

15. What will be the output of the following code?

```
tup1 = (1,2,[1,2],3)
tup1[2][1]=3.14
print(tup1)
```

(SQP 2021)

a) (1,2,[3.14,2],3)
b) (1,2,[1,3.14],3)
c) (1,2,[1,2],3.14)
d) Error Message

Ans: b) (1,2,[1,3.14],3)

`tup1[2][1]` accesses the value of the second element in the third element of the tuple.

Since the third element of the tuple is a list, this operation will be valid as lists are mutable.

This will change the value from `2` to `3.14`

So the new tuple will be `(1,2,[1,3.14],3)`

Chapter 1 MCQs

1. Which of the following is an invalid identifier?

a) _name
b) name1
c) 1name
d) nam1e

2. Which of the following is a valid identifier?

a) 123age
b) @ge
c) ag-e
d) ag_e

3. What type of token is `1`

a) identifier
b) keyword
c) literal
d) punctuator

4. Which of the following is not a punctuator?

a) :
b) #
c) +
d) =

5. Which of the following is equivalent to True?

a) 1
b) "Hello"
c) [1]
d) All of the above

6. Which of the following is data type of `x` in the following code?

```
a = 10
a = "hello"
x = a
```

a) int
b) float
c) string
d) Error

7. Find the respective values of a,b and c:

```
>>> a,b,c = 100, 200, [300]
```

a) 100, 200, 300
b) (100, 200, [300])
c) 100, 300, 200
d) 100, 200, [300]

8. Find the output of `print("A", "B","C", sep="#", end="#")`

a) A#B#C#
b) A#B#C
c) A B C#
d) None of the above

9. Consider the declaration `abc = 100,200,300`**. What is the data type of** `abc`

a) int
b) list
c) tuple
d) string

10. Which of the following is not a core data type?

a) class
b) int
c) tuple
d) list

11. In Python, memory address values can be classified as which data type?

a) string
b) int
c) float
d) None of the above

12. Which of the following is immutable?

a) list
b) dictionary
c) set
d) tuple

13. Which of the following operators/functions does not return an `int`**?**

a) //
b) int()
c) id()
d) /

14. What is the output of `(3**4)/(3)`

a) 3
b) 27.0
c) 3.0
d) 27

15. Which of the following is associated with complex numbers?

a) i
b) j
c) k
d) l

16. Which of the following methods will create a deep copy of a list?

a) copy.copy()
b) list()
c) slicing
d) All of the Above

17. Evaluate `5**2*3`

a) 75
b) 390625
c) 15625
d) 40

18. Evaluate `True and False or True`

a) True
b) False
c) Error
d) None of the above

19. Evaluate `1 and True or False and True`

a) True
b) False
c) 1
d) None of the above

20. Evaluate `3%4**5 + 100`

a) 0
b) 100
c) 103
d) None of the above

21. Evaluate `8/4/2//1`

a) 1
b) 1.0
c) 4.0
d) 4

22. Which of the following operators as highest precedence?

a) <
b) not
c) and
d) or

23. Which of the following is the correct output of the following code?

```
x = 0
while True:
    print(x, end="")
    x+=1
    if x == 8:
        break
```

a) 012345678
b) 01234567
c) 12345678
d) 1234567

23. Which of the following is the correct output of the following code?

```
x = 10
if x/2 == 5:
    if x% 5 == 0:
        print(True)
    else:
        print(False)
else:
    print("false")
```

a) True
b) False
c) false
d) None of the above

24. What is the output of the following code?

```
x = 10
while x > 2:
    print(x, end="")
    x -= 1
```

a) 1098765432
b) 109876543
c) 98765432
d) 9875643

25. What is the output of the following code?

```
for i in range(10, 2, -3):
    print(i, end="")
```

a) 1098765432
b) 109876543
c) 1074
d) 10741

Questions 26 to 28 are based on this given list

```
L=["hello",[1,2,3],8,"potato"]
```

26. Find the output of `L[1]`

a) `8`
b) `[[1,2,3]]`
c) `"hello"`
d) `[1,2,3]`

27. Find the output of `L[1][2]`

a) `2`
b) `3`
c) `e`
d) `l`

28. Find the output of `L[1:2]`

a) `[[1,2,3]]`
b) `[1,2,3]`
c) `[1,2,3],8`
d) `[[1,2,3],8]`

29. Find the output

```
s="Programming is fun!"
print(s[4:8])
```

a) grammi
b) gramm
c) rammi
d) ramm

30. Let a list T be defined as `T=[10,20,(30,40,50),60]`. What will be the value of T after executing `T[2][1]=100`?

a) `(10,20,(100,40,50),60)`
b) `(10,20,(30,100,50),60)`
c) Error
d) None of the above

31. Which of the following is an invalid identifier name?

a) h$llo
b) _hello
c) true
d) none

32. Find the output of the following code

```
l1=[1,2,3]
l2=l1
l3=list(l1)
print(id(l1)==id(l2))
print(id(l1)==id(l3))
```

a) `True`
`True`
b) `False`
`True`
c) `True`
`False`
d) `False`
`False`

33. Find the output of the following code

```
l1=["Welcome","to","Programming"]
l2=l1
l3=list(l1)
l2[1]="for"
l3[1]="with"
print(l1,l2,l3)
```

a) `["Welcome","with","Programming"] ["Welcome","with","Programming"] ["Welcome","with","Programming"]`
b) `["Welcome","for","Programming"] ["Welcome","for","Programming"] ["Welcome","with","Programming"]`
c) `["Welcome","to","Programming"] ["Welcome","for","Programming"] ["Welcome","with","Programming"]`
d) `["Welcome","to","Programming"] ["Welcome","to","Programming"] ["Welcome","with","Programming"]`

34. If there is a set defined as `set1={1,2,3,1,4,8,2,9,3}`, what will you see when you print it?
a) `{1,2,3,1,4,8,2,9,3}`
b) `{1, 2, 3, 4, 8, 9}`
c) `[1,2,3,1,4,8,2,9,3]`
d) `(1,2,3,1,4,8,2,9,3)`

35. Which of the following error would you get if you try to change the value of a item in a immutable datatype?
a) No error will be given
b) `IndentationError`
c) `ValueError`
d) `TypeError`

36. What error will be thrown if you execute `True/False`?
a) `SyntaxError`
b) `ZeroDivisionError`
c) `NameError`
d) None of the above

37. What will be the output?

```
x=0
while x<=10:
    x+=1
    if x==5:
        continue
    if x==8:
        break
    print(x,end="")
print(" finished")
```

a) `123467 finished`
b) `12345678910 finished`
c) `1234567 finished`
d) `12345678 finished`

38. Evaluate `False and (not True) or (False and True)`
a) `True`
b) `False`
c) `None`
d) `Error`

39. Evaluate `(2*5**2- 5//2)/6`
a) `8`
b) `7`
c) `8.0`
d) `7.0`

40. Which of the following are not operators used with strings?
a) in
b) +
c) %
d) *

41. What will be the value of `l1` **if** `l1=list("hello")` **?**
a) `["hello"]`
b) `["h","e","l","l","o"]`
c) `["he","llo]`
d) `"hello"`

42. What will be the output of `(1,2,3)<(1,2,1,9)`
a) `True`
b) `False`
c) `Error`
d) None of the above

Questions 43 to 46 are based on this given code:

```
t=(1,2,["a","b3"],(2,"ok"))
```

43. What is `t[1]`?

a) `["a","b3"]`

b) `1`

c) `2`

d) `(1,2)`

44. What is `t[2][1][-1]` ?

a) `3`

b) `'3'`

c) `'b'`

d) `'b3'`

45. What is `t[3][1][0]`?

a) `(2,"ok")`

b) `"o"`

c) `'2'`

d) `2`

46. What will be the output?

```
a=2
while a>=0:
    print(a,end="")
    a+=1
```

a) 23456789

b) 23456789....infinitely

c) 22222222....infinitely

d) No output

47. What are the outputs of `2 or 9` , `0 or 4` , `3 and 6` respectively?

a) `9`, `0`,`3`

b) `2`,`4`,`3`

c) `2`, `4`,`6`

d) `9`,`0`,`6`

48. What will be the output of this code

```
s="hello welcome to python"
print(s.count("e",2))
```

a) `1`

b) `2`

c) `3`

d) `4`

49. Which of these is not a type of token in python ?

a) comments

b) keywords

c) literals

d) operators

50. What is the datatype output of the `split` function in strings?
a) tuple
b) list
c) string
d) dictionary

Chapter 1 MCQ Answers

Q	A	Q	A	Q	A	Q	A	Q	A
1	C	11	B	21	B	31	A	41	B
2	D	12	D	22	B	32	C	42	B
3	C	13	D	23	A	33	B	43	C
4	C	14	B	24	B	34	B	44	B
5	D	15	B	25	C	35	D	45	B
6	C	16	D	26	D	36	B	46	A
7	D	17	A	27	B	37	A	47	C
8	A	18	A	28	A	38	B	48	B
9	C	19	A	29	D	39	C	49	A
10	A	20	C	30	C	40	C	50	B

Chapter 1 Expression Evaluation Questions

1. `2+(3**2)//5 + 4/2`
a) 5.0
b) 5
c) 4
d) 4.0

2. `1 and 2 or 0`
a) 1
b) 2
c) 0
d) 3

3. `True or False and not True`
a) `True`
b) `False`
c) `None`
d) Error

4. `2*3**2 + (3/2)*4`
a) 25.0
b) 24
c) 25
d) 24.0

5. `0 or 3 and 4`
a) 0
b) 3
c) 4
d) 5

6. `True or not False and False`
a) `True`
b) `False`
c) `None`
d) Error

7. `(2**4) +2 0//3 +3*5`
a) 36
b) 37.0
c) 37
d) 36.0

8. `(1 or 2) and (5 and 3)`
a) 1
b) 2
c) 5
d) 3

9. `4**2 + 3/1 + 10//3`
a) 23
b) 22
c) 23.0
d) 22.0

10. `True and not False and False`
a) `True`
b) `False`
c) `None`
d) Error

Chapter 1 Expression Evaluation Answers

Q	A	Q	A	Q	A	Q	A	Q	A
1	A	2	B	3	B	4	D	5	C
6	A	7	C	8	D	9	D	10	B

Chapter 1 Output Finding Questions

1. Find output

```
s="hELlo pYtHON @123"
new=""
for i in s:
    if i.islower():
        new+=i.upper()
    elif i.isupper():
        new+=i.lower()
    elif i.isdigit():
        new+="$"
    else:
        new+="&"
print(new)
```

a) `HelLO$$$PyThon&$`
b) `hELlo&PytHON$$&&&`
c) `HelLO&PyThon&&$$$`
d) None of the above

2. Find output

```
s="wElcOMe tO pYThon 101"
new=""
for i in range(len(s)):
    if s[i].isupper():
        new+=s[i].lower()
    elif s[i].islower():
        new+=s[i+1]
    else:
        new+="#"
print(new)
```

a) `wEecOom PytHon ####`
b) `EecOom #Oo#Yyton ####`
c) `WEec#### #Oo#Yyton`
d) None of the above

3. Find output

```
s1="hElLo tHerE"
s2=""
for i in range(len(s1)-2):
    if s1[i].isupper():
        s2+=s1[i].lower()
    if s1[i].islower():
        s2+=s1[i+1]
    if s1[i].isalpha():
        s2+=s1[i+2]
    else:
        s2+="#"
print(s2)
```

a) `'EeLl #Hhr'`

b) `EleLLol  t#HehrrE`

c) `hEllo# tHEre`

d) None of the above

4. Find output

```
s1="hElLo tHerE"
s2=""
for i in range(len(s1)-2):
    if s1[i].isupper():
        s2+=s1[i].lower()
    elif s1[i].islower():
        s2+=s1[i+1]
    elif s1[i].isalpha():
        s2+=s1[i+2]
    else:
        s2+="#"
print(s2)
```

a) `'EeLl #Hhr'`

b) `EleLLol  t#HehrrE`

c) `hEllo# tHEre`

d) None of the above

5. Find output

```
s1="PErCUsSIOnISTS"
s2=""
s3=""
for i in range(0,len(s1),2):
    if s1[i].isupper():
        s2+=s1[i+1]
    elif s1[i].islower():
        s2+=s1[i]+s1[i+1]
for j in range(0,len(s2)):
    if s2[j].islower():
        s3+="#"
    else:
```

```
        s3+=s2[j]
print(s3)
```

a) `ErCsInSS`
b) `#R#CS#NSS`
c) `EINSST`
d) `E#C#I#SS`

Chapter 1 Output Finding Answers

Q	A	Q	A	Q	A	Q	A	Q	A
1	C	2	B	3	B	4	A	5	D

Functions

Functions are blocks of code with a **specific purpose**

The advantages of using functions in Python programming are:

- They **increase readability**, particularly for longer codes, since a function can be called at any point in the program.
- They **reduce code length**, since the same code is not repeated in different places.
- They allow **reuse of code**.
- They increase **modularity** of the code and make it easier to maintain.

Types of Functions

Built-in Functions

- These are the functions that are ready-made in python, and are frequently used in many programs.
- These do **not** need to be imported

Such functions include `print()`, `range()`,`input()`, etc.

```
print("Hello world")
input("How are you?")
for i in range(5):
    print("hello")
```

Module Functions

- The python standard libraries consists of collections functions other than inbuilt ones, which can be used through **modules**
- Modules are a **group of functions** that can be imported into a program by the import statement and can be used accordingly.
- Modules have a `.py` extension
- In fact, when you make a file such as `code.py` you are making a module!

Ways to Import Modules

When we import a module, the contents of the module are considered as an object and we set an identifier for these objects. The identifier can be the name of the module or we can also set an **alias.**

- `import module_name`

  To call a function in a module imported this way, you must write the **name of the module** before it

  For example

  ```
  import math
  print(math.sqrt(16)) #This will print the square root of 16
  ```

The output of this code will be `4,0`

- `import module_name as alias`
  In this method, you can import a module with any name you want and you can use that name before calling a function

  This new name is known as an **alias** and it points to the module object in the memory**.**

  For example:

```
import random as ran
print(ran.randint(2,5)) # This will print a random integer between 2 and 5
```

- `from module_name import function1, function2, ...`

  In this method, you don't need to specify the modules name** before calling a function, since you will be importing specific functions from it

  For example:

```
from math import abs, sin
print(abs(-5)) # This will print the absolute value of -5
print(sin(0))  # This will print the sine of 0 radians
```

  The output of the two functions will be `5` and `0` respectively

  To import **all the functions** from a python module, you can simply use an * instead of the function names

```
from math import *
print(floor(3.2))  # Calculates the floor value of 3.2
print(abs(-10))    # Calculates the absolute value of -10
print(sin(pi/6))   # Calculates the sine of π/6 radians
```

```
The output for the functions will be `3`, `10` and `0.499` respectively
```

User Defined Functions

These are functions the **user can define** and **use** in their programs.

For example, the following function adds two numbers a and b:

```
def sum(a,b):
    print(a + b)

sum(2,3)
```

The output of this code will be `5`

Creating User Defined Functions

To create user defined functions, you have to use the `def` keyword and then writing the name of the function. It is recommended to use a name which is relevant to what the function is actually doing

The general syntax is as follows:

```
def identifier(parameter1, parameter2, ...): # header
    '''Docstring'''                          # body starts here
    <statement>
    <statement>

<code outside function>
```

Let's break down the components of a function:

1. **Header:** The header is the first line in the function definition. It begins with the `def` keyword and ends with a colon `:`. It includes the following:
    1. **Identifier:** The name of the function. It follows the same naming conventions as variables.
    2. **Parameter(s):** Placeholders for values passed during a function call.
2. **Body:** The body consists of code and documentation of the function. It comprises of the following:
    1. **Docstring:** A multiline string containing important information about the purpose of the function. The purpose of the function is conventionally stated in the first line and all other details regarding parameters, usage, etc. are written in the following lines.
    2. **Statements:** Lines of code which are executed at the function call.

For example, if we want to define a function to print "hello" 5 times, we do the following:

```
def say():
    '''Prints hello 5 times.'''
    for i in range(5):
        print("Hello")
```

Every time the function `say()` is called, "Hello" is printed five times.

If you want to use this in **another python file**, you will have to import it using one of the methods above. Although you must remember to store both the files in the **same memory location.** The file containing the functions you have defined will behave like a module.

Arguments and Parameters

In the `range()` function, we call it with certain values.

```
for i in range(2,10,2):
    print(i)
```

The values we *passed* within the brackets of the `range()` function call are called **arguments.**

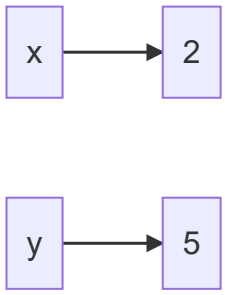
Arguments are the **values given to a function** at the time of the function call.

If we want a user defined function to take arguments, we have to define it using **parameters**.

Parameters can be thought of as **place holders** of values that should be given while calling the function.

Let's look at the example below to understand:

```
def sum(x,y):   # here x and y are the parameters
    print(x+y)

sum(2,5)        # here 2 and 5 are the arguments
```



Here `x` and `y` take the values `2` and `5` respectively when the function is called.

The parameter behaves as variable which only that particular function can access. When we pass an argument during a function call, you can thingkof it as assigning a variable a value.

Arguments v/s Parameters

	Parameter	Argument
1	These are values used while defining the functions as placeholders for values to be passed at the time of the function call	These are the actual values that are passed to the function at the time of the function call
2	Also called "Formal Parameters" or "Formal Arguments"	Also called "Actual Parameters" or "Actual Arguments"

Types of Arguments

The following are the types of arguments supported by python:

- **Positional Arguments**

  These arguments are values which are respectively assigned to parameters in the order the parameters have been set in the function definition.

  For example:

  ```
  def sum(a,b)
      print(a+b)

  sum(3,7) # a is assigned 3 and b is assigned 7
  ```

  The output will be `10`.

- **Default Argument**

In a default argument, you can give the default value to a parameter while defining a function which it will take **if the value is not passed** while calling the function

For example:

```
def sum(a,b=3):
    print(a+b)

sum(2)   # Here we did not pass the value of b, so a=2 and b=3
sum(2,5) # Here we did pass the value of b, so a=2 and b=5
```

The output of this code will be `5` and `7` respectively

- **Keyword Argument**

  These are the values you can give to parameters **while calling the function** by specifying the parameter you are passing a value for.

  The special thing here is that the **order of the keyword arguments does not matter** as long as the **identifiers** are correct

```
def decrement(a,b,c):
    a -= 1
    b -= 1
    c -= 1
    print(a,b,c)

decrement(b=10,c=8,a=5)   # order does not matter
```

The output of this code will be `4 9 7`

- **Arbitrary Arguments** (args)

  These are used when we do not know **how many** arguments our function is supposed to receive.

  Whatever arguments are received are stored in a tuple.

  Example:

```
def increment(*num): # the * indicates that num will store arbitrary args
    for i in num:
        print(i + 1 ,end=" ")

increment(2,3,4,5)
increment(3,5)
```

The output of this code will be `4 5 6 7` and `4 6` for the 2 function calls respectively

- **Keyword Arbitrary Arguments** (kawargs)

  These are used to store an unknown number of keyword arguments but with each argument having keywords assigned to them. The keyword arguments are stored in the form of a dictionary.

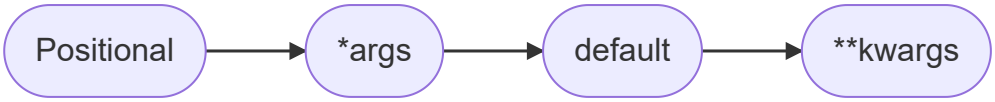
```
def show_details(**names): # the ** indicates that names will store keyword args
    print(names["name"], names["age"])

show_details(name="qprogramming", age=20)
```

As you can see, we pass keyword arguments at the function call. Keep in mind that the dictionary stores the data as:

```
{"<keyword1>":value, "<keyword2>":value}
```

Mixing Types of Arguments

When you have different kinds of arguments in a function call, the order of arguments will be (from left to right):



For example:

```
def function(a,*b,c=10,**d):
    print(a)
    print(b)
    print(c)
    print(d)

function(1,2,3,4,5,c=15,first=200,second=250)
```

Output:

```
1
(2, 3, 4, 5)
15
{'first': 200, 'second': 250}
```

Returns

We use the `return` statement in functions when we want the function to output a value but not necessarily display it. When we return an object in a function, the data stored in the object can be assigned to a variable or used in an operation.

You can think of a return value of the function giving us an output on which we can perform different operations.

For example, let's define a function to possess a Boolean value depending on whether a given number n is prime or not

```
def isprime(n):
    for i in range(2,n):
        if n%i==0:
            return False
    return True
```

As soon as the program encounters the return statement, it **immediately stops** and outputs the value next to it.

When you call a function, the return value can be treated as an object in Python.

For example, using the function `isprime(n)` we want to create a list of values out of the list `[3,4,5,6,7,8,9,10]` which are prime

```
numbers=[3,4,5,6,7,8,9,10]
new=[]
for i in numbers:
    if isprime(i):  # isprime() will return either True or False
        new.append(i)
print(new)
```

So the output of this code will be `[3,5,7]`.

By default, a function returns `None` if no return value is given.

Multiple Returns

We just saw how to use the `return` statement and how we can make functions return values

We can also make it return multiple values, which you **must** separate by commas.

In the end, the function returns a `tuple` of all the values

For example:

```
def test(a,b):
    return a+1, b+1

print(test(2,3))
print(test(5,6))
```

The output of the above code will be `(3,4)` and `(6,7)` respectively for the 2 function calls, and both of these are **tuples**.

Flow of Execution

Let us understand the flow of execution in functions using the example below:

```
def sum(a,b):
    print(a+b)

print("before")
sum(2,3)
print("after")
```

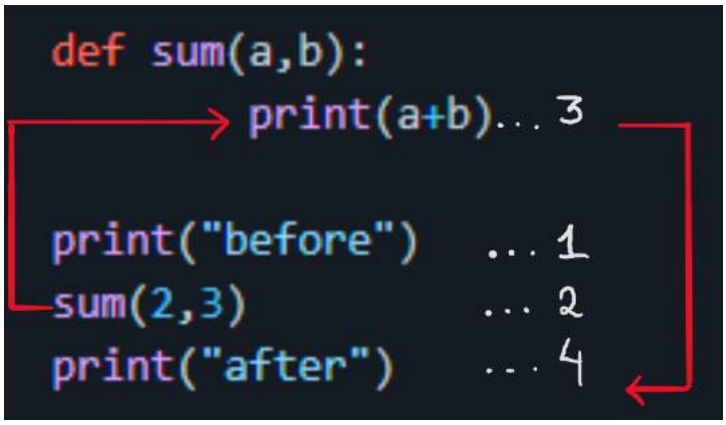
The lines **before the function call** are executed first, then the lines in the **function definition** and then finally the lines **after the function call**.

When the program is run, the Python interpreter will read it line by line. When it encounters a function definition, it will take note of it, but skip it and execute the code within it **only when the function is called**.

So the output of the above code will be as follows:

```
before
5
after
```

We can understand it through the logical diagram below:



Scope of a Variable

While defining variables inside a function, one has to keep in mind what **scope** it should have.

The scope of a variable tells the interpreter which parts of a program can access it.

There are 2 kinds of scopes:

Local Scope	Global Scope
When you directly define a variable inside a function, it has a local scope. It can only be accessed inside the function	When a variable is defined with global scope, its value can be accessed and both outside and inside the function
No keyword required to change a local variable within a function	`global` keyword required to change global variable within a function

Let's understand this through an example:

```
def test():
    x=5
    print(x)

x=7
print(x)
test()
print(x)
```

In this code, the variable x defined as `x=5` inside the `test()` function is a **local variable**. So it's not assigning the value `5` to `x`, it is instead creating a local variable x with value 5.

So the output of this code will be:

```
7
5
7
```

If we want to make x global, we just have to use the `global` keyword:

```
def test():
    global x
    x=5
    print(x)

x=7
print(x)
test()
print(x)
```

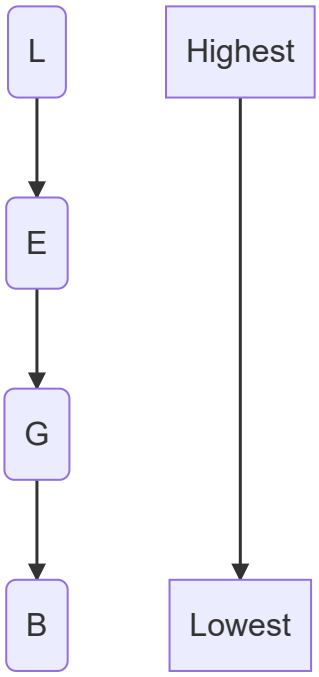
Now, `test()` will be able to modify the value of `x` since it is treated as a **global variable** within the function.

Hence the output of this code will be

```
7
5
5
```

LEGB Rule

The LEGB rule is a rule used to determine the precedence of variable scopes. It stands for **L**ocal **E**nclosing **G**lobal **B**uilt-In.



Thus, a local variable definition will take precedence over a global variable definition as seen in the following example:

```
x = 5
def display():
    x = 7
    print(x)

display()
```

The output will be `7` since the local declaration `x=7` has higher precedence than the global declaration `x=5`,

Mutable Data Types as Arguments

What happens if we pass a mutable data type as an argument, for example a list? Let's look at an example:

```
def modify(list1):
    list1.append(1)

listx = [2,3,4,5]
modify(listx)
print(listx)
```

The output will be `[2,3,4,5,1]`. Why? It's simple. A function can directly modify a variable with a mutable data type. So, if we pass a list through a function, the function will be able to modify the list directly.

But, this will not work with immutable data types:

```
def modify(number):
    number = 5

n = 6
modify(n)
print(n)
```

In this case, the out put will be `6` since the function cannot modify the value of n, as integers are immutable.

However, do note that reassigning the value of a variable, even if it contains a mutable data type, will not work as we **can only modify the existing value**.

```
def change(list1):
    list1 = [2,4]

l = [1,2,3]
change(l)
print(l)
```

The output will be `[1,2,3]` as the value of `l` will not be affected if we try to reassign it. Here, we just created a local variable `list1`.

Chapter 2 Solved Examples

1. Which of the following components are part of a function header in Python?

(SQP 2021)

a) Function Name
b) Return Statement
c) Parameter List
d) Both a and c

Ans: d) Both a and c

In the function header we add the name and parameter list as follows:

```
def func_name(parameter1, parameter2,...):
    function body
```

2. Which of the following function header is correct?

(SQP 2021)

a) def cal_si(p=100, r, t=2)
b) def cal_si(p=100, r=8, t)
c) def cal_si(p, r=8, t)
d) def cal_si(p, r=8, t=2)

Ans: d) def cal_si(p, r=8, t=2)

In the function header, default arguments are always on the right hand side and can never be before a positional argument.

3. Which of the following is the correct way to call a function?

(SQP 2021)

a) my_func()
b) def my_func()
c) return my_func
d) call my_func()

Ans: a) my_func()

This is the only correct way to call a function `my_func`.

4. What will be the output of the following code?

```
def my_func(var1=100, var2=200):
    var1+=10
    var2 = var2 - 10
    return var1+var2
print(my_func(50),my_func())
```

(SQP 2021)

a) 100 200
b) 150 300
c) 250 75
d) 250 300

Ans: d) 250 300

For the function call `my_func(50)`, `var1` is assigned the value of `50`.

Therefore we will get `var1 = 60` and `var2 = 190`, so the sum is `250`.

For the function call `my_func()`, `var1 = 110`, `var2 = 300`

So we will get the sum as `300`.

So the correct option will be `250 300`

5. What will be the output of the following code?

```
value = 50
def display(N):
    global value
    value = 25
    if N%7==0:
        value = value + N
    else:
        value = value - N

print(value, end="#")
display(20)
print(value)
```

(SQP 2021)

a) 50#50
b) 50#5
c) 50#30
d) 5#50#

Ans: b) 50#5

The first `print` will output `50#`

When we call `display(20)`, the value of `value` changes to 5

Therefore the output of the second `print` will be `5` and the final output is `50#5`.

6. What will be the output of the following code?

```
x = 3
def myfunc():
    global x
    x+=2
    print(x, end=' ')
print(x, end=' ')
myfunc()
print(x, end=' ')
```

(SQP 2021)

a) 3 3 3
b) 3 4 5
c) 3 3 5
d) 3 5 5

Ans: d) 3 5 5

Before the first `print`, `x=3` so the output will be `3`.

When we call `my_func` the value of x changes to 5 and it gets printed.

The final `print` also prints 5 so the final output will be `3 5 5`.

7. What is the output of the following code snippet?

```
def ChangeVal(M,N):
    for i in range(N):
        if M[i]%5 == 0:
            M[i]//=5
        if M[i]%3 == 0:
            M[i]//=3

L = [25,8,75,12]

ChangeVal(L,4)
for i in L:
    print(i,end="#")
```

a) 5#8#15#4#
b) 5#8#5#4#
c) 5#8#15#14#
d) 5#18#15#4#

Ans: b) 5#8#5#4#

Let's look at how the function will modify the list.

1. Since 25 is divisible by 5, it will be changed to 5
2. Since 8 is not divisible by 3 or 5 it will remain the same
3. Since 75 is divisible by 5 it will be changed to 15 then since 15 is divisible by 3 it will be changed to 5
4. Since 12 is divisible by 3 it will be changed to 4

Therefore the new list is `[5,8,5,4]` and thus the output will be `5#8#5#4`.

8. A binary file employee.dat has following data

Empno	empname	Salary
101	Anuj	50000
102	Arijita	40000
103	Hanika	30000
104	Firoz	60000
105	Vijaylakshmi	40000

```
def display(eno):
    f=open("employee.dat","rb")
    totSum=0

    try:
        while True:
            R=pickle.load(f)
            if R[0]==eno:
                __________ #Line1
            totSum=totSum+R[2]
    except:
```

```
        f.close()

    print(totSum)
```

When the above mentioned function, display (103) is executed, the output displayed is 190000. Write appropriate jump statement from the following to obtain the above output.

(SQP 2021)

a) jump
b) break
c) continue
d) return

Ans: c) continue

We know the output will be `190000`. When we look at the code within the loop, we see that in each iteration, the salary is being added to the sum of salaries in `totSum`. However, a special case will arise when the employee number is `103`. If we add all the salaries of all employees, we get the value `220000`. What is the relation between `220000` and `190000`? Simple, they differ by `30000` which is equal to the salary of employee 103. This means that 103's salary is not being added so in that particular iteration, the line `totSum=totSum+R[2]` is not being executed. The only appropriate jump statement for this purpose would be `continue` since it skips all the remaining code and continues to the next iteration.

9. What will be the output of the following code snippet?

```
var = 100
def change1():
    var = 105

def change2():
    global var
    var = 105

print(var, end=" ")
change1()
print(var, end=" ")
change2()
print(var, end="")
```

a) 100 100 100
b) 105 105 105
c) 100 100 105
d) 105 105 100

Ans: c) 100 100 105

The first `print` prints the first value of `var` which is `100`.

When `change1` is executed, no change occurs in the global `var` variable, so the second `print` also prints `100`.

When `change2` is executed, the value of `var` changes to `105` which is thus outputted by the third `print` statement.

10. Fill in the blank line with suitable code so that the output is 5

```
var = 10

def change():
    ________ # blank line
    var = var - 5

change()
print(var)
```

a) var = 10
b) var
c) local var
d) global var

Ans: d) global var

The `global` keyword allows us to change the value of `var`.

Chapter 2 MCQs

1. Which of these is *not* a way to import the math module?

a) `from math import *`
b) `import math`
c) `importing math`
d) `import math as m`

2. Which of the following is an inbuilt function?
a) output()
b) range()
c) sin()
d) randint()

3. Which of the following keywords is used to begin the function definition?
a) void
b) return
c) int
d) def

4. In the following function definition, what is c?

```
def (a,b,c=10):
    print(a+b+c)
```

a) Argument
b) Default Parameter
c) Keyword Parameter
d) Arbitrary Parameter

5. Which of the following is false?
a) The variables declared inside a function are by default global variables
b) Default parameters go from right to left in the function definition
c) A function can return multiple values in the form of a tuple
d) Functions from modules like math, statistics, etc. can only be used once the module is imported

6. What is the output of the following code?

```
def test():
    x=10
    x+=2
x=5
test()
print("After calling, the value of x is",x)
```

a) After calling, the value of x is 10
b) After calling, the value of x is 12
c) After calling, the value of x is None
d) Error

7.Which of the following is the correct way to define default parameters?

a) `def func(a,b=10,c=12)`

b) `def func(a=1,b,c=5)`

c) `def func(a=2,b,c)`

d) `def func(a,b=10,c)`

8. What will be the output?

```
def calculate(a,b,c):
    return a*10,b*20,c*30
calculate(1,2,3)
```

a) `[10,40,90]`

b) `(10,40,90)`

c) `[10,20,30]`

d) `(20,30,40)`

9. If there is no return statement in a function, by default it will return:

a) Void

b) 0

c) Null

d) None

10. Which of the following is not a feature of Python functions?

a) Increase code length

b) Increases readability

c) Reusability of code

d) Makes it easy to locate errors

11. Name the statement that sends back a value from the function

a) def

b) return

c) input

d) print

12. Which of these is the term for describing where a variable can be accessed in a program?

a) designation

b) area

c) local

d) scope

13. Determine the output of this code:

```
def func():
    print("hello",end="")
print("hi",end="")
func()
print("python",end="")
func()
```

a) hi hello python hello
b) hello hi python hello
c) hi hello hello python
d) hello hi hello python

14. Functions not returning any values are called:
a) Void functions
b) User defined functions
c) Inbuilt functions
d) Modular functions

15. Find the ouput

```
def add(a,b=10):
    return a+b
x=add(10)
y=add(15,20)
print(x+y)
```

a) 35
b) 45
c) 55
d) 65

16. Which of the following are not present in the function header?
a) return value
b) name of function
c) `def` keyword
d) list of parameters

17. If multiple values are used with `return` statement, which datatype is returned?
a) list
b) string
c) dictionary
d) tuple

18. Which of the following is the correct way to call a function called `function_name()`?
a) call function_name()
b) function_name()
c) execute function_name()
d) run function_name()

19. What symbol do we use to denote that the function should be receiving arbitrary arguments?
a) $
b) %
c) *
d) #

20. What will be the output of this code?

```
def func():
    l1.append(4)
    l2=[1,2,3,4]
l1=[1,2,3]
l2=[1,2,3]
func()
print(l1,l2)
```

a) `[1,2,3,4] [1,2,3,4]`
b) `[1,2,3] [1,2,3,4]`
c) `[1,2,3] [1,2,3]`
d) `[1,2,3,4] [1,2,3]`

21. Arbitrary arguments are stored as which datatype?
a) string
b) list
c) tuple
d) dictionary

22. Which of the following can be edited by a function?
a) list
b) tuple
c) dictionary
d) a and c

23. Which of the following statements is true?
a) A function written by a programmer as per their requirements is called inbuilt function
b) A local variable having the same name as a global variable hides the global variable in the function
c) Python modules have a `.dat` extension
d) Tuples can be modified by a function

24. Identify which line(s) have error in this code? (with respect to versions after and including python 3)

```
def minus(a,b)      #....(i)
    final = a - b   #....(ii)
    print final     #....(iii)
    return final    #....(iv)
```

a) lines i and iv
b) only line i
c) lines i and iii
d) only line iv

25. Suppose you have a variable x defined outside a function. You want to change the value of x inside the function. Which statement should be used?
a) No statement required; it can be done directly
b) `local x`
c) `return x`
d) `global x`

Questions 26-30 are assertion-reason based, answer using these options:
a) Both A and R are true, R explains A
b) Both A and R are true, R does not explain A
c) A is true but R is false
d) A is false but R is true

26. A: The variables defined inside functions are called global variables
R: Variables defined inside a function can only be used by that function

27. A: Default arguments are defined in the function header for certain parameters
R: They are used in case a certain value is not passed in the function call

28. A: Arbitrary arguments are represented by an asterix (*)
R: Arbitrary arguments are used when you are unsure if the value will be passed in the function call

29. A: Keyword arguments are values given during the function call, where you write the name of the parameter and give it a value
R: The order of keyword arguments does not matter

30. A: If a function is defined at the top of a program, it will be executed before any other part of the program
R: The program has a very specific flow of execution

31. Which of the following function calls is not legal based on the following function definition?

```
def func(a,b,c=20,d=15):
    print(a+b+c+d)
```

a) `func(2,5,d=3)`
b) `func(2,b=3,d=5)`
c) `func(2,c=3,d=3)`
d) `func(2,3,4)`

32. Which of the following is the correct name resolution rule in Python?
a) Local, Enclosing, Global, Built-in
b) Local, Enclosing, Built-in, Global
c) Enclosing, Local , Global, Built-in
d) Enclosing, Local , Built-in, Global

33. Find the output

```
def sq(*num):
    for i in num:
        print(i*i,end=" ")
sq(2,3,4,5,6)
```

a) 4
9
16
25
36
b) 4 9 16 25 36

c) 49162536
d) (4,9,16,25,36)

34. What datatype is used in random.shuffle()?
a) string
b) dictionary
c) list
d) tuple

35. Which of the following code should be used in the function definition to complete the code so that the result of the code is `10`?

```
def f(n):
    #Function definition
print(f(5))
```

a) `return n*2`
b) `return 'n'*2`
c) `print(n*2)`
d) `print('n'*2)`

36. Which of the following are not inbuilt functions?
a) `print()`
b) `input()`
c) `eval()`
d) `log()`

37. Find output

```
def f():
    x=5
    global y
    y=y-5
    print(x,y)

x=10
y=20
print(x,y)
f()
print(x,y)
```

a) `10  20`
`5 15`
`10 15`
b) `10  20`
`5 15`
`5 15`
c) `20  10`
`15 5`
`10 15`
d) `20  10`

`15 15`

`10 20`

38. A collection of related functions in a python file which may be imported into programs is called

a) collection

b) module

c) inbuilt functions

d) group

39. Find output

```
def func():
    d1["D"]=60
    d2={"A":30,"B":40,"C":50,"D":60}
d1={"A":30,"B":40,"C":50}
d2={"A":30,"B":40,"C":50}
func()
print(d1,d2)
```

a) `{"A":30,"B":40,"C":50,"D":60} {"A":30,"B":40,"C":50,"D":60}`

b) `{"A":30,"B":40,"C":50} {"A":30,"B":40,"C":50,"D":60}`

c) `{"A":30,"B":40,"C":50,"D":60} {"A":30,"B":40,"C":50}`

d) `{"A":30,"B":40,"C":50} {"A":30,"B":40,"C":50}`

40. Find output

```
def f(*a):
    print(type(a),end=",")
    print(a)
f(2)
```

a) `<class 'int'>,2`

b) `<class 'tuple'>,(2,)`

c) `<class 'int'>,(2,)`

d) `<class 'tuple'>,2`

41. Identify the false statement

a) Mutable datatypes such as dictionaries, lists can be modified by a function

b) Default parameters may or may not be passed during the function call

c) Inbuilt functions can be used after using the `import` statement to import the library

d) Keyword arguments can be written in any order, irrespective of the order in the function definition.

42. Which of the following are correct ways to import the module `math` in python?

i. `import math` ii. `importing math` iii. `from math import *` iv. `from math import all` v. `import math as med`

a) ii and iv only

b) i, ii and v

c) i ,iii and v

d) i,ii and iv

43. Choose the correct function header

a) `def func(a,b=10,c)`

b) `def func(a,b=20,c=30)`

c) `def func(a=10,b,c)`

d) `def func(a=10,b=20,c)`

44. Determine the correct flow of execution of the following code

```
def func(a,b):                #....A
    for i in range(a,b):      #....B
        print(i)              #....C
x,y=10,20                     #....D
func(x,y)                     #....E
print("Execution finished")   #....F
```

a) D➡E➡A➡B➡C➡F

b) A➡B➡C➡D➡E➡F

c) D➡A➡B➡C➡E➡F

d) A➡B➡C➡E➡D➡F

45. If we want a function to send back a value to the function call, but not necessarily print it, what statement should be used?

a) `def`

b) `pass`

c) `void`

d) `return`

46. Find output

```
def repeat(a,b):
    print(a*b,end=" ")
repeat("Hello",5)
repeat(5,3)
```

a) 15 HelloHelloHelloHelloHello

b) Hello Hello Hello Hello Hello 15

c) HelloHelloHelloHelloHello 15

d) Hello15Hello15Hello15Hello15Hello15

47. Which of these is an advantage of functions?

a) They make the code less readable

b) They reduce code length

c) They create repetition of code

d) They do not allow for modularity

48. Which of these does the help() function not do?

a) It gives instructions to the user on how to use the function

b) It gives the information stored in the functions definitions docstring

c) It gives the user help on how to use the function

d) It executes the function

49. Find output

```
def f(a,b,c=5):
    print(a+b-c,end=" ")

f(2,c=8,b=5)
f(2,3)
```

a) `15 10`
b) `-1 0`
c) `0 -1`
d) `10 15`

50. Find output

```
def f1():
    global x
    x=x*2
    print(x,end=" ")
def f2():
    x=5
    x=x+2
    print(x,end=" ")
x=3
print(x,end=" ")
f1()
f2()
print(x)
```

a) `3 7 6 6`
b) `3 6 7 7`
c) `3 6 7 6`
d) `3 7 7 7`

Chapter 2 MCQ Answers

Q	A	Q	A	Q	A	Q	A	Q	A
1	C	11	B	21	C	31	C	41	C
2	B	12	D	22	D	32	A	42	C
3	D	13	A	23	B	33	B	43	B
4	B	14	A	24	C	34	C	44	A
5	A	15	C	25	D	35	A	45	D
6	C	16	A	26	D	36	D	46	C
7	A	17	D	27	A	37	A	47	D
8	B	18	B	28	C	38	B	48	D
9	D	19	C	29	B	39	C	49	B
10	A	20	D	30	D	40	B	50	C

Chapter 2 Output Finding Questions

1. Find output of the following code

```
def Change(a,b=10):
    a=a*b
    b=a//b
    print(a,"$",b)
    return a
x=20
y=30
x=Change(x,y)
print(x,"$",y)
y=Change(y)
```

a) `600 $ 30`
`600 $ 30`
`300 $ 30`

b) `600 $ 20`
`600 $ 30`
`300 $ 30`

c) `300 $ 20`
`600 $ 30`
`600 $ 30`

d) `600 $ 20`
`300 $ 30`
`600 $ 30`

2. Find output of the following code

```
def alter(s):
    new=""
    for i in s:
        if i.islower():
            new+=i.upper()
        elif i.isdigit():
            new+="k"
        else:
            new+="#"
    print(new)

alter("Hello, welcome to programming 101!")
```

a) kELLO##WELCOME#TO#PROGRAMMING#kkk#
b) HELLO##WELCOME#TO#PROGRAMMING#kkk#
c) #ELLO##WELCOME#TO#PROGRAMMING#kkk#
d) kELLOkkWELCOMEkTOkPROGRAMMINGk###k

3. Find output of the following code

```
def change(s):
    new=""
    for i in s:
        if i.isupper():
            new+=i.lower()
        elif i.isdigit():
            new+="b"
        else:
            new+="$"
    print(new)

change("PYTHoN1234@com")
```

a) pyth$nbbbb$$$$
b) Pythonbbbb$$$$
c) pythonbbbb@com
d) pyth$n$$$$bbbb

4. Find output of the following code

```
def calc(a,b,c=30):
    a=a+b
    b=b+c
    c=c+(b//a)
    print(a,"#",b,"#",c)
    return a
x=5
y=10
z=15
x=calc(x,y)
print(x,"#",y,"#",z)
z=calc(x,y,z)
print(x,"#",y,"#",z)
```

a) `15 # 40 # 32`
`25 # 10 # 15`
`25 # 25 # 32`
`15 # 10 # 10`

b) `15 # 40 # 32`
`15 # 10 # 15`
`25 # 25 # 16`
`15 # 10 # 25`

c) `15 # 20 # 16`
`15 # 10 # 15`
`25 # 15 # 16`
`15 # 5 # 25`

d) `25 # 20 # 32`
`15 # 10 # 16`
`15 # 25 # 15`
`15 # 10 # 32`

5. Find the output of the following code

```
def func():
    global x
    x=x+2
    y=x+10
    print(x,"@",y)
x=5
y=15
print(x,"@",y)
func()
print(x,"@",y)
```

a) `7 @ 15`
`7 @ 17`
`7 @ 17`

b) `5 @ 17`
`7 @ 15`
`7 @ 15`

c) `5 @ 15`
`7 @ 17`
`7 @ 17`
d) `5 @ 15`
`7 @ 17`
`7 @ 15`

Chapter 2 Output Finding Answers

Q	A	Q	A	Q	A	Q	A	Q	A
1	B	2	C	3	A	4	B	5	D

Chapter 2 Case Study

1. A student learning python has just learnt about the quadratic formula. He is trying to make a code where he takes in the values of a,b and c from the user and finds the roots of the quadratic equation

$$ax^2 + bx + c \tag{2}$$

Help him complete the task successfully

```
from ____ import sqrt                #Statement 1
def quad(a,b,c):
    D=b**2-4*a*c
    if D<0:                          #Statement 2
        return "Roots are not real"
    sq_D=_____(D)                    #Statement 3
    x1=(-b+sq_D)/2*a
    x2=(-b-sq_D)/2*a
    return x1,x2

print(type(quad(2,3,-4)))             #Statement 4
print(quad(3,c=5,b=2))               #Statement 5
```

i. What should be in the blank of statement 1?
a) statistics
b) math
c) cmath
d) either b or c

ii. Is it correct to continue the code under the `if` block marked as statement 2?
a) Yes, because the `return` statement will allow execution of code under it
b) No, because the rest of the code *must* be in an `else` block corresponding to it
c) Yes, because the `return` statement returns the value and doesn't execute any lines under it
d) No, because the execution of code stops at the return statement.

iii. What should be in the blank of statement 3?

a) `math.sqrt`

b) `sqrt`

c) `cmath.sqrt`

d) None of the above

iv. What is the output of statement 4?

a) `<class 'dictionary'>`

b) `<class 'string'>`

c) `<class 'list'>`

d) `<class 'tuple'>`

v. What kind of arguments are used in statement 5?

a) Keyword

b) Default

c) Arbitrary

d) Normal

vi. Is the code in statement 5 syntactically correct?

a) Yes, because the order does not matter in keyword arguments

b) No, because the order must be the same as defined in the parameters

c) Yes, because the keyword arguments must begin from left

d) No, because the parameter `a` will not get any value

2. The CEO of a data science company wants to create a function to update employee salaries. She calls you, a python expert, to help her complete the code

```
def update(_________________):                    #Statement 1
    for i in people:
        if i[0]==_______:                         #Statement 2
            i[1]=________                         #Statement 3
    print("Updating finished")

people=[["Rob",5000],["Carla",5500],["Louise",6000]]
update("Rob",5500)                              #Statement 4
update("Carla")                               #Statement 5
print(people)                                   #Statement 6
```

i. In statement 1, the CEO wants there to be 2 parameters in the function, firstly "name" and then "new_salary". The parameter "new_salary" should take the value of 5000 if no value is given to it. Which of the following function headers is appropriate for this situation?

a) `def update(name,new_salary)`

b) `def update(new_salary=5000,name)`

c) `def update(name,new_salary=5000)`

d) `def update(new_salary,name=5000)`

ii. She wants to loop through all the records and update when the correct record is found by name. What should be in the blanks of statements 2 and 3 respectively?

a) `name`, `new_salary`

b) `new_salary`, `name`

c) `name`, `5000`
d) `"Rob"`, `5000`

iii. Is this method of updating logically correct?
a) No, since list is a immutable datatype
b) Yes, because the list "people" is local to the function
c) No, because the list "people" must be made global in the function
d) Yes, because mutable datatypes can be updated through functions

iv. What will be the value of "new_salary" in statements 4 and 5 respectively?
a) `5000`, `6000`
b) `5000`, `5500`
c) `5500`, `5000`
d) `6000`, `5000`

v. What will be the output of statement 6?
a) `[["Rob",5000],["Carla",5500],["Louise",6000]]`
b) `[["Rob",5500],["Carla",5000],["Louise",6000]]`
c) `[["Rob",5000],["Carla",5000],["Louise",6000]]`
d) `[["Rob",5500],["Carla",5500],["Louise",6000]]`

vi. What kind of argument is executed by the function call in statement 5?
a) Normal
b) Keyword
c) Arbitrary
d) Default

Chapter 2 Case Study Answers

Case	i.	ii.	iii.	iv.	v.	vi.
Case Study 1	B	C	B	D	A	A
Case Study 2	C	A	D	C	B	D

Python Standard Library Modules & Functions

Built In Functions

Simple Functions

Function	Use	Syntax	Example
`print()`	To output values	`print(*args, sep=<string>, end=<string>)`	`print(2,3, sep="#", end="")`
`input()`	To input values and return a string of the values	`input(<string>)`	`name=input("Enter name:")`
`len()`	To get the length of a sequence	`len(object)`	`len([1,2,3])`
`max()`	To find the maximum value out of a sequence	`max(<sequence>)`	`max([2,3,4])` `max("abc")`
`min()`	To find the minimum value out of a sequence	`min(<sequence>)`	`min([2,3,4,5])` `min("xyz")`
`sum()`	Find the sum of the values of a sequence with numeric data	`sum(<sequence>)`	`sum([2,3,4,5])`

String Methods

Method	Use	Syntax	Example
`len()`	Returns the length of the string	`len(<string>)`	`>>> len("hello")` 5
`.title()`	Returns string with first letter of each word in uppercase and rest in lowercase	`<string>.title()`	`>>> "heLlo tHerE".title()` 'Hello There'
`.lower()`	Returns string with all uppercase characters converted to lowercase	`<string>.lower()`	`>>> "hello THERE".lower()` 'hello there'
`.upper()`	Returns string with all lowercase characters converted to uppercase	`<string>.upper()`	`>>> "hello THERE".upper()` 'HELLO THERE'
`.capitalize()`	Returns a string with the first letter capitalized.	`<string>.capitalize()`	`>>> "hello there".capitalize()` 'Hello there'
`.find()`	Returns the index position of a substring and if not found, then -1	`<string>.find(<substring>)`	`>>> "hello there".find("lo")` 3
`.index()`	Returns the index of a substring and produces a `ValueError` if not found.	`<string>.index(<substring>)`	`>>>"hello world".index("wo")` 6
`.isalnum()`	Returns `True` if the characters of the string are either alphabets or numeric. If any other characters are found(symbols,spaces,etc) it returns `False`	`<string>.isalnum()`	>>>"hellothere".isalnum() True >>>"hi there".isalnum() False
`.isalpha()`	Returns `True` if the characters of the string are alphabets	`<string>.isalpha()`	>>>"hello".isalpha() True >>>"hello!".isalpha() False
`.isdigit()`	Returns `True` if the characters of the string are digits(numeric values)	`<string>.isdigit()`	>>>"12345".isdigit() True >>>"hello123".isdigit() False
`.isspace()`	Returns `True` if the string is non-empty and all characters are white spaces(blank, tab, newline, carriage return)	`<string>.isspace()`	>>>" \n \t".isspace() True >>>"python ".isspace() False
`.islower()`	Returns `True` if the string has all lowercase characters or lowercase characters and other non-alphabet characters	`<string>.islower()`	>>>"hello there!".islower() True >>>"Hey!".islower() False
`.isupper()`	Returns `True` if the string has all uppercase characters or uppercase characters and other non-alphabet characters	`<string>.isupper()`	>>>"HEY PYTHON".isupper() True >>>"hellO therE".isupper() False
`.istitle()`	Returns `True` if the string is in title case(the first letter of every word in the string is uppercase and rest lowercase)	`<string>.istitle()`	>>>"Hello There!".istitle() True >>>"hello How".istitle() False
`.swapcase()`	Makes lowercase characters uppercase and vice-versa	`<string>.swapcase()`	>>>"hEllo tHERe".swapcase() HeLLO TherE
`.split()`	Returns a list of words delimited by a specified substring. If no delimiter is given, then it is space by default	`<string>.split(<delimiter>)`	>>>"Programming is fun".split() ["Programming","is","fun"] >>>"Learning coding!".split("g") ['Learnin', ' codin', '!']

Method	Use	Syntax	Example
`.partition()`	Partitions a string into 3 parts based on the occurrence of a substring: 1. Substring before seperator 2. Seperator 3. Substring after seperator and stores it in a tuple	`<string>.partition(<substring>)`	>>>"Hello there how are you?".partition("how") ('Hello there ', 'how', ' are you?')
`.count()`	Returns the number of times a substring occurs in a given string. If no start and end values are specified, it searches the whole string	`<string>.count(<substring> ,start,end)`	>>>"Python is really really fun!".count("really") 2 >>>"Python is really really fun!".count("really",15,28) 1

`.lstrip`, `.rstrip()`, `.strip()`	Returns string after removing the spaces on the left(lstrip),right(rstrip) or both(strip)	`<string>.lstrip()`
		`<string>.rstrip()`
		`<string>.strip()`
			>>>" Ok then".lstrip()
			"Ok then"
			>>>" python ".strip()
			"python''

`.startswith()`	Returns `True` if the given string starts with the given substring else `False`	`<string>.startswith(<substring>)`	>>>"Hello".startswith("He") True >>>"Python".startswith("he") False
`.endswith()`	Returns `True` if the given string ends with the given substring else `False`	`<string>.endswith(<substring>)`	>>>"Python".endswith("on") True >>>"Hello".endswith("py") False

List Methods

Method	Use	Syntax
`.append()`	To add an **element** to the end of the list	`<list>.append(<object>)`
`.extend()`	To add the **elements** of one list to the end of another	`<list>.extend(<list>)`
`.pop()`	To remove an element based on an index (by default -1)	`<list>.pop(<index>)`
`.index()`	To find the index of a particular element. If not found, raises `ValueError`	`<list>.index(<object>)`
`.insert()`	To insert an element at a particular index	`<list>.insert(<object>, <index>)`
`.remove()`	To remove a particular element	`<list>.remove(<object>)`
`.clear()`	To clear all elements in a list	`<list>.clear()`
`.count()`	To count the number of occurrences of an element in a list	`<list>.count(<object>)`
`.sort()`	To sort a list (by default ascending)	`<list>.sort(reverse=<bool>)`
`.reverse()`	To reverse the order of elements in a list	`<list>.reverse()`

Dictionary Methods

Method	Use	Syntax
`.keys()`	To get an iterable with the keys	`<dict>.keys()`
`.values()`	To get an iterable with the values	`<dict>.values()`
`.items()`	To get an iterable containing key-value pairs as tuples	`<dict>.items()`
`.pop()`	To remove a particular key-value pair	`<dict>.pop(<key>)`
`.clear()`	To clear all key-value pairs	`<dict>.clear()`
`.get()`	To get the value associated with a key	`<dict>.get(<key>)`
`.update()`	To add elements of another dictionary to a particular dictionary	`<dict>.update(dict2)`

`os`

The `os` module is used for functions related to the operating system:

Function	Use	Syntax	Example
`os.getcwd()`	To get path of current working directory as string	`os.getcwd()`	`cwd = os.getcwd()`
`os.chdir()`	To change path of current working directory	`os.chdir(<path>)`	`os.chdir("./records")`
`os.mkdir()`	To make a new directory	`os.mkdir(<path>)`	`os.mkdir("./students/class11")`
`os.rmdir`	To remove a directory	`os.rmdir(<path>)`	`os.rmdir("./students/class11")`
`os.remove()`	To remove a file	`os.remove(<path>)`	`os.remove("./students/class11/file.txt")`
`os.path.isfile()`	To check if a file exists (boolean)	`os.path.isfile(<path>)`	`os.path.isfile("./students/rollnos.csv")`

`datetime`

The `datetime` module is used for functions related to date and time

Function	Use	Syntax	Example
`datetime.datetime.now()`	To get the current date and time	`datetime.datetime.now()`	`>>> print(datetime.datetime.now())` 2021-09-21 18:09:51.757310

`random`

The `random` module is used to generate pseudo-random numbers.

Function	Use	Syntax	Example
`random.random()`	To generate float value within the range 0.0 and 1.0	`random.random()`	`>>> random.random()` 0.6648913465425406
`random.randint()`	To generate random integer between x and y (both inclusive)	`random.randint(x,y)`	`>>> random.randint(1,6)` 5
`random.randrange()`	To generate random integer between x and y (y not inclusive)	`random.randrange(x,y)`	`>>> random.randrange(2,6)` 3
`random.choice()`	To generate a randomly selected element from a specified sequence	`random.choice(<sequence>)`	`random.choice([2,3,4,5]` `random.choice("welcome")`
`random.sample()`	To generate a list with k random items from a sequence	`random.sample(<sequence>,k)`	`random.sample([1,2,3,4,5],k=2)` `random.sample("abcd",k=3)`

`statistics`

The `statistics` module is used for calculating statistics of numerical data.

Function	Use	Syntax	Example
`statistics.mean()`	To get the mean of values in a sequence	`statistics.mean(<list of nos.>)`	`statistics.mean([1,2,3,4])`
`statistics.mode()`	To get the mode of values in a sequence	`statistics.mode(<list of nos.>)`	`statistics.mode([1,2,2,3.4])`
`statistics.median`	To get the median of values in a sequence	`statistics.median(<list of nos.>)`	`statistics.median([1,2,3,4])`

`math`

The `math` module provides mathematical functions defined by the C standard, which can only be used with **real numbers**.

Function	Use	Syntax	Example
`math.sin()`	Sine of x in radians (returns float)	`math.sin(x)`	`>>> math.sin(0)` 0
`math.fabs()`	Absolute value of x. x may be float or integer but returns float	`math.fabs(x)`	`>>> math.fabs(-4.3)` 4.3
`math.sqrt()`	Square root of x x may be float or integer but returns float	`math.sqrt(x)`	`>>> math.sqrt(64)` 8.0
`math.pow()`	x^y (x to the power of y), returns float	`math.pow(x,y)`	`>>> math.pow(4,3)` 64.0
`math.factorial()`	Factorial of x, returns int	`math.factorial(x)`	`>>> math.factorial(4)` 24
`math.floor()`	Floor value of x Greatest integer less than or equal to x, returns int	`math.floor(x)`	`>>> math.floor(3.4)` 3
`math.ceil()`	Ceiling value of x Smallest integer more than or equal to x, returns int	`math.ceil(x)`	`>>> math.ceil(3.4)` 4
`math.gcd()`	GCD(Greatest Common Divisor) of x and y, returns int	`math.gcd(x,y)`	`>>> math.gcd(24,10)` 2

`math.fmod()`	x%y with sign of x
	x and y may be float or integer, returns float
		`math.fmod(x,y)`	`>>> math.fmod(9.8,4.9)`
			0.0
			`>>> math.fmod(4,4.9)`
			4.0

`math.pi`	constant value of π	`math.pi`	

Chapter 3 Solved Examples

1. The return type of the input() function is

(SQP 2021)

a) string
b) integer
c) list
d) tuple

Ans: a) string

The `input()` function always returns a string.

2. Identify the output of the following Python statements.

```
lst1 = [10, 15, 20, 25, 30]
lst1.insert(3, 4)
lst1.insert(2, 3)
print(lst1[-5])
```

(SQP 2021)

a) 2
b) 3
c) 4
d) 20

Ans: b) 3

The first `insert` will modify the list to `[10, 15, 20, 4, 25, 30]`

The second `insert` will modify the list to `[10, 15, 3, 20, 4, 25, 30]`

Therefore `lst1[-5] = 3`.

3. What will be the output of the following code?

```
import random
List=["Delhi","Mumbai","Chennai","Kolkata"]
for y in range(4):
    x = random.randint(1,3)
    print(List[x],end="#")
```

(SQP 2021)

a) Delhi#Mumbai#Chennai#Kolkata#
b) Mumbai#Chennai#Kolkata#Mumbai#
c) Mumbai# Mumbai #Mumbai#Delhi#
d) Mumbai# Mumbai #Chennai#Mumbai

Ans: b) Mumbai#Chennai#Kolkata#Mumbai#

The value of `x` can be either 1, 2 or 3 as per the return values of `random.randint()`.

This means that the output can only contain Mumbai, Chennai and Kolkata, which allows us to eliminate options a and c.

Now, if you look at the print statement, it contains the argument end="#" which means that a # will be printed at the end of each value.

This allows u to eliminate d, leaving us with b as the answer.

Chapter 3 MCQs

1. Which of the following functions does not belong to the math module?

a) abs
b) fabs
c) pow
d) sin

2. Which of the following functions does not belong to the random module?

a) randint
b) randrange
c) randnum
d) random

3. Which of the following modules would you use to get the current date and time?

a) datetime
b) time
c) os
d) random

4. What name do we give to functions that come with Python?

a) Python functions
b) Defined functions
c) Built-in functions
d) User defined functions

5. All Python modules have which extension?

a) .cpp
b) .python
c) .txt
d) .py

6. What is the use of the help function?

a) To give us tips to code better
b) To give documentation of different modules, functions, etc.
c) To help us become good programmers
d) To debug our code

7. What will be the range of numbers that can be outputted by the following code:

```
import random

n = random.randint(1,5)
print(n)
```

a) 0-4 (0 and 4 included)
b) 0-5 (0 included)
c) 1-5 (1 included)
d) 1-5 (1 and 5 included)

8. Which of the following modules is not part of the Python Standard Library?

a) statistics
b) random
c) numpy
d) time

9. Which of the following modules is part of the Python Standard Library?

a) datetime
b) matplotlib
c) pandas
d) scipy

10. The module used to deal with operating system functions is?

a) opsys
b) os
c) Os
d) OS

11. What will be the output of the following code?

```
import math

num = -5
print(math.fabs(num))
```

a) 0
b) -5
c) 5
d) 5.0

12. What will be the output of the following code?

```
import statistics

lst = [1,2,3,4,5]
print(statistics.median(lst))
```

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

13. What is the return type of the len function?

a) int
b) str
c) float
d) None of the above

14. What is the return type of the abs function?

a) int
b) float
c) complex
d) Depends on the input data type

15. What is the return type of the sum function?

a) int
b) float
c) complex
d) Depends on the data type in the sequence

16. Which of the following is not associated with lists?

a) .clear()
b) del
c) .pop()
d) .keys()

17. Which of the following can be used to always split strings into a tupof 3 elements?

a) .partition()
b) .split()
c) .splice()
d) .break()

18. Which of the following functions is used to get the value of e

a) math.e()
b) math.e
c) math.exp
d) math.exp(2)

19. Which of the following is the output of `mode([1,2,3,1])`

a) 0
b) 1
c) 2
d) 3

20. Which of the following is the name of the collection of built in functions and modules?

a) Library
b) Python Library
c) Python Standard Library
d) Python Built In Library

Chapter 3 MCQ Answers

Q	A	Q	A	Q	A	Q	A	Q	A
1	A	5	D	9	A	13	A	17	A
2	C	6	B	10	B	14	D	18	B
3	A	7	D	11	D	15	D	19	B
4	C	8	C	12	C	16	D	20	C

Chapter 3 Output Finding Questions

1. What will be the output of the following code?

```
import random

lst = ["A", "B", "C", "D", "E"]

for i in range(3):
    n = random.randint(0,3)
    print(lst[n], end="#")
```

a) A#B#E#
b) A#B#D
C) A#B#C#
d) A#E#C#

2. What will be the output of the following code?

```
name = "qprogramming"
name = list(name)

name.pop(2)
print("#".join(name))
```

a) q#p#r#o#g#r#a#m#m#i#n#g
b) q#p#o#g#r#a#m#m#i#n#g
c) q#p#r#o#g#r#a#m#m#i#g
d) q#p#o#g#r#a#m#m#i#n#g#

3. What will be the output of the following code?

```
dict1 = {"name":"raj", "age":17}
dict1['profession'] = "student"
dict1[age] += 1

for key, value in dict1.items():
    if key == list(dict1.keys())[0]:
        print(key, value)
```

a) age 17
b) name raj
c) profession student
d) age 18

4. What will be the output of the following code?

```
string = "qprogramming"
string += "1234"

tot = 0
for ch in string:
    if ch.isalnum():
        tot += 1

print(tot == len(string))
```

a) 16
b) False
c) True
d) Error

5. What will be the output of the following code?

```
from math import fabs
from random import randint

n = -5.2
print(randint(0,fabs(n)))
```

a) Error
b) 5
c) 2
d) 1

Chapter 3 Output Finding Answers

Q	A	Q	A	Q	A	Q	A	Q	A
1	C	2	B	3	B	4	C	5	A

Chapter 4: File Handling

Advantages of Files

- They allow us to store data permanently
- They allow us to reuse data
- They allow us to access data quickly
- They allow us to easily manipulate data

Types of Files

Text Files

- Stores data as binary, but we see decoded text
- Can be opened in text editor eg. Notepad
- Each line has an end of line (EOL) character (default is \n)
- Contents separated by whitespace, comma or tab (\t)
- Examples: `.txt` files, `.py` files
- Python has **inbuilt** file object methods to work with such files (`read()`, `write()`, `.readlines()`, etc.)

Binary Files

- Stores data as binary and original raw form
- A single change to the binary file can corrupt it
- Difficult to remove errors since we cannot understand it
- Not readable by humans
- `.dat` , `.bin` file extension
- We use the `pickle` module in Python to work with such files

CSV Files

- One type of text file
- Includes multiple rows known as **records**
- Each element in a row is separated by a comma (or a different **delimiter)**
- `.csv` file extension
- We use the `csv` module in Python to work with such files

CSVs can be represented as tables as well:

	Field 1	**Field 2**	**Field3**
Record 1	R1, F1	R1, F2	R1, F3
Record 2	R2, F1	R2, F2	R2, F3

Differences Between Files

Factor	Text	Binary	CSV
Use	Stores sequence of unicode characters	Stores large files such as images and videos as the raw bytes	Stores data in the form of a table with records (rows) and columns (attributes) and each element in a record is separated by a delimiter (by default a comma)
Extension	.txt, .py	.dat, .bin	.csv
Human Readability	Readable	Not Readable	Readable
Python Support	Inbuilt methods and functions	pickle module	csv module
Speed	Slower	Faster	Slower
EOL	Every line ends with EOL which is `\n` by default	No EOL	Terminates a line automatically when the delimiter is not used after data.

Opening a File

To open a file we need to use the **open** function. Python has several functions for handling files in the `io` module.

```
file = open(path, mode)
```

This assigns a file object to the variable `fiie`. A file object, also known as **file handle**, is simply a reference to a file on a machine.

There are two major arguments we see here: `path` and `mode`. Let's take a deeper look at them.

Types of Paths

Absolute Path

The path starting from the **drive** (C:, D:, E:, etc)

Let's say we have a set of folders in the order:

```
C: > user > qprogramming > documents > file.txt
```

The absolute path to this file would be `C:\\user\\qprogramming\\documents\\file.txt`

We can also use single `\` instead of `\\`, but in Python we may come across a case wherein a single dash plus a character could represent an escape sequence.

Example:

```
"C:\users\qprogramming\documents\total.csv"
```

We know that `\t` is used for a tab space so it might cause an error if we use such a path in our string, we can put an "r" before our string which tells the interpreter to only use the **raw** string and ignore special sequences:

```
r"C:\users\qprogramming\documents\total.csv"
```

Relative Path

The relative path of a file is the path relative to the current working directory.

To use a relative path we have to keep in mind two symbols:

1. Single Dot ("."): Represents the current working directory (CWD)
2. Double Dot (".."): Represents the directory containing the current working directory (parent directory)

Let's understand with the help of an example:

If we are in the directory:

```
C:\users\qprogramming\documents\
```

Our CWD is this path and we represent it by `.`

The path to the parent directory is `C:\users\qprogramming` and we represent it by `..`

So, if we want to use a file `test.txt` located in the directory `C:\users\qprogramming\documents\files` directory, we use the path `./files/test.txt` to access the file.

If the file is located in `C:\users\qprogramming\` , we can use the path `../test.txt` to access the file.

Access Modes

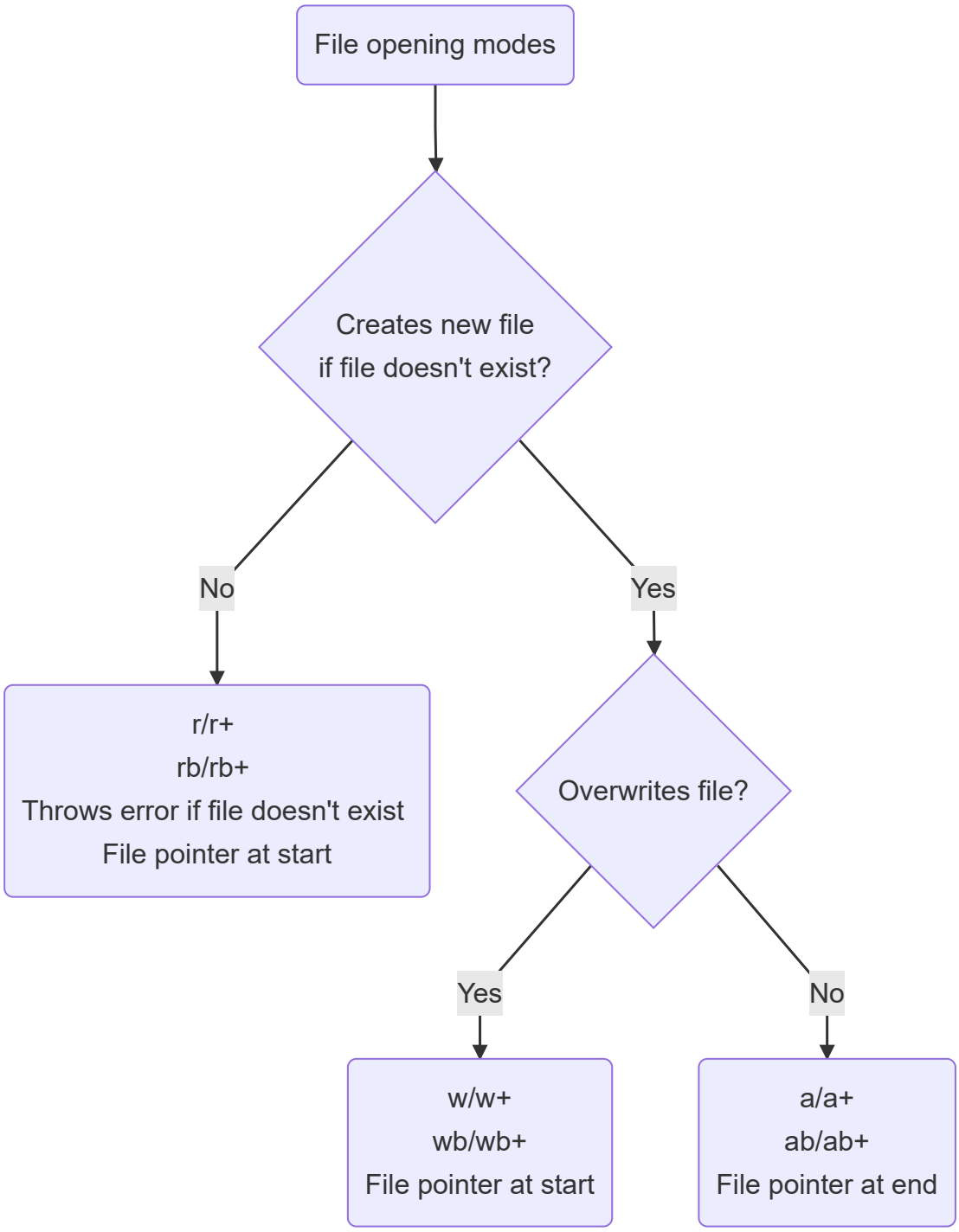
- Optional argument
- By default it is read mode (r)
- Access mode is added as a string

S. No.	Mode	Binary Equivalent	Purpose	Creates New File	Offset
1	r	rb	Read-only mode	No	Beginning
2	r+ or +r	rb+ or r+b	Read and write mode	No	Beginning
3	w	wb	Write Mode	Yes	Beginning
4	w+ or +w	wb+ or w+b or +wb	Write and Read Mode	Yes	Beginning
5	a	ab	Append mode	Yes	End
6	a+ or +a	ab+ or a+b or +ab	Read and Append Mode	Yes	End

All append and write modes (regular and binary) create a new file if the given file name/path does not exist. If you try to open a file which does not exist in read mode, you will get a `IOError`.

If you add the name of the file directly, the program will search for the file in the **current working directory/present working directory**.

Fun Fact: you can open text files in binary mode since they are stored in the binary format



Attributes of a file object

```
file.closed # True if file has been closed else False
file.mode # Access mode in which file was opened
file.name # Name of file
```

Closing a File

- `.close()` method

```
file.close()
```

`with` clause

- **Advantage:** don't have to close file as it is automatically closed after the code within the `with` has been executed
- Memory efficient
- Simpler syntax

Syntax

```
with open(filename, mode) as file:
    <code>
```

Here, the `file` variable is still assigned a file object.

In conclusion, to open a file we need to specify its path as well as the mode we want to open it in.

Text Files

Writing to a File

`.write()` method

- Writes string to text file
- To go to a new line, we need to add "\n" at the end of each line in the string

Let's make a file `file.txt`:

```
with open("file.txt", "w") as file:
    string = "Hello World!\nHow are you?"
    file.write(string)
```

When we run the code, we get the following output in `file.txt`:

```
Hello World!
How are you?
```

In reality, `.write()` writes to a buffer and the content is only added to the file when the file is closed.

You should also known that this method returns the number of bytes written to a file.

`.writelines()` method

- Writes multiple strings
- Have to pass iterable like list, tuple, string

Let's create the same file again with this method:

```
with open("file.txt", "w") as file:
    lines = ["Hello World!\n", "How are you?"]
    file.writelines(lines)
```

When we run the code, we get the following output in `file.txt`:

```
Hello World!
How are you?
```

Note that you have to add \n at the end of every line barring the last one, if you want each element in the list to go to a newline.

Key Point: `.write()` returns number of characters written, `.writelines()` does not.

Reading a File

- Using r, r+, w+, a+ mode

`.read()` method

- Reads a specified number of bytes of data
- **Syntax:** `file.read(n)`, where `n` is the number of bytes to read.
- In ascii, **each character is stored as one byte**, so n also represents **number of characters**

Example:

```
with open("file.txt", "r") as file:
    content = file.read(5) # first five bytes of data
```

If a **negative** number or no number is given for `n`, it reads *all* of the data:

```
with open("file.txt", "r") as file:
    content = file.read() # all data
```

`.readline(n)` method

- Reads `n` characters in one complete line until end of line, indicated by \n character

```
with open("file.txt", "r") as file:
    content = file.readline(10) # first 10 characters
    print(content)
```

When we run the code the output will be `Hello Worl` since those are the first 10 bytes in the file

If no argument is provided, the entire line is read.

```
with open("file.txt", "r") as file:
    content = file.readline()
    print(content)

# if first line is "Hello World!", the output would be "Hello World!" and a blank line due to \n
```

`.readlines()` method

- Reads all lines with \n and returns a **list** of strings of each line
- \n at end of each string in the list
- If we give a number as argument, it acts as a character count.

Let's suppose we have the following file `file.txt`:

```
Hello World!
How are you?
```

We create the following program:

```
with open("file.txt", "r") as file:
    content = file.readlines()
    print(content)
```

The output will be:

```
['Hello World!\n', 'How are you?']
```

We can split the lines using `.split()` to get individual words:

Using the file in previous example, we can print output line by line using `.splitlines()`

```
with open("file.txt", "r") as file:
    content = file.readlines()

    for line in content:
        print(line.split())
```

The output will be:

```
['Hello', 'World!']
['How', 'are', 'you?']
```

Using the file in previous example, we can print output line by line using `.splitlines()`

```
with open("file.txt", "r") as file:
    content = file.readlines()

    for line in content:
        print(line.splitlines())
```

The output will be:

```
['Hello World!']
['How are you?']
```

Program to display properly line by line

```
with open("file.txt", "r") as file:
    content = file.readlines()

    for line in content:
        print(line.strip("\n")) # removes trailing \n
```

The output will be:

```
Hello World!
How are you?
```

Setting Offsets

Offsets help us use the position of the pointer in the file.

`.tell()`

Specifies the position in the file

```
with open("file.txt", "r") as file:
    print(file.tell()) # returns byte position
```

The output will be `0` as read mode opens the file at the start.

`.seek()`

Moves pointer to particular position

```
file.seek(<offset>, <reference point>)
```

The reference point can be:

0: beginning
1: current position
2: end of file

The default is 0

Example

We have a file `file.txt` with the following contents:

```
Hello World!
How are you?
```

Let's run the following code:

```
with open("file.txt", "r") as file:
    content = file.read() # moves cursor to end of file when finished
    print(file.tell())

    file.seek(0) # moves to beginning of file
    print(file.tell())

    file.seek(10) # moves to 10th byte from beginning
    print(file.tell())

    file.seek(0,2) # moves to end of file (0 bytes from end)
    print(file.tell())
```

We will get the following output:

```
26
0
10
26
```

Binary Files

Binary File Operations

- **Serialization** is the process of **transforming** data or an object in the memory (RAM) to a stream of **bytes** called **byte streams**.
- These byte streams in a binary file can then be stored in a disk or in a database or sent through a network.
- Serialization process is also called **pickling**.
- **De-serialization** or **unpickling** is the inverse of pickling process where a byte stream is **converted** back to a Python object.

`pickle` Module

`load()`

- Used to unpickle/deserialize
- Syntax: `pickle.load(file)`

```
import pickle

with open("file.dat", "rb") as file:
    data = pickle.load(file)
```

The `data` variable is assigned the data stored in the binary file and it can be a list or tuple etc. in the format Python stores them as.

If there are multiple objects stored together, for example, `[1,2,3,4][4,5,6]`

We can use `while` to get all the data:

```
import pickle

with open("file.dat", "rb") as file:

    # runs until end of data
    try:
        data = pickle.load(file)
        while data:
            print(data)
            data = pickle.load(file) # try to load file

    # EOFError occurs when file ends but we try to retreive data from it
    except EOFError:
        print("Done Reading")
```

The output of this program will be:

```
[1, 2, 3, 4]
[4, 5, 6]
Done Reading
```

Each time the loop runs, the program loads an object until the loop terminates.

`dump()`

- Used to pickle
- Syntax: `pickle.dump(object, file)`

```
import pickle

with open("file.dat", "wb") as file:
    data = [1,2,3,4]
    pickle.dump(data, file)
```

If we load the data in file.dat and assign to variable data, `data = [1,2,3,4]`:

```
import pickle

with open("file.dat", "rb") as file:
    data = pickle.load(file)
    print(data)
```

Output: `[1,2,3,4]`

Append

Let's say we want to make a binary file with multiple records. To accomplish this we will use the binary file in append mode, `ab`.

Let's say I want to append the following records for a student database:

```
{"roll": 1, "name": "A", "marks": 95}
{"roll": 2, "name": "B", "marks": 92}
{"roll": 3, "name": "C", "marks": 81}
{"roll": 4, "name": "D", "marks": 85}
```

I can write the following program to input the data for each record and append individual records to our binary file. Remember, binary files can store all kinds of data such as lists, tuples, dictionaries, etc.

```
import pickle

with open("students.dat", "ab") as file:
    to_continue = "y" # variable to check if you want to continue adding data
    while to_continue == "y":
```

```
        # record dictionary
        record = {}

        # fields to fill
        rollno = int(input("Enter Roll Number: "))
        name = input("Enter Student Name")
        marks = int(input("Enter Marks:"))

        record["roll"] = rollno
        record["name"] = name
        record["marks"] = marks

        pickle.dump(record, file)

        # check if user would like to continue entering data
        to_continue = input("Would you like to enter more data? If yes, enter y,
else n: ")
```

Now, enter the data and it will be appended to the file.

Here are the steps we used:

1. Create file in append mode
2. Create input fields
3. Append the object for each record to the binary file

Search

In the last example, we added records to a student database. Let's now see how we can access data of a particular student.

We know our binary file stores multiple objects and we also know how to load multiple objects.

Let's make a program to print the names of students scoring above 90 in the exam:

```
import pickle

with open("students.dat", "rb") as file:

    # we use exception handling to catch errors in loading data
    while True:
        try:
            record = pickle.load(file)

            if record['marks'] > 90:
                print(record['name'])
        except EOFError:
            pass
```

Output:

```
A
B
```

We can use this method to get any kinds of data we want.

Here are the steps we use:

1. In a try - except block, load the file once
2. If the file is not empty, loop through each object and check the required condition
3. When the file ends, exit the loop and program

The following is another example, where we make a function to get the details of a student by their roll number:

```
import pickle

def get_details(rno):
    with open("students.dat", "rb") as file:

        # we use exception handling to catch errors in loading data
        while True:
            try:
                record = pickle.load(file)
                if record['roll'] == rno:
                    print("Name:",record['name'], "Marks:",record['marks'])
                    return # exit function

            except EOFError:
                print("Roll no. not in database")

get_details(2)
get_details(5)
```

The output will be:

```
Name: B Marks: 92
Roll no. not in database
```

Update

Let's say we now want to update the data for a record.

We will follow the following method:

1. Find record to update
2. Make changes in loaded record
3. Write back to the the file at the location of the record

Here is where our offsets come into play. Before we load the next object, we can get the current location using `file.tell()`. This will give the starting location of the next object. We store this position and then, if we come across the record to change, we set our cursor to the starting point using `file.seek()` and then overwrite the previous data. The following example will make it easier to understand:

Let's add 2 marks to student C's score:

```
import pickle

with open("students.dat", "rb+") as file:
    while True:

        try:
            pos = file.tell() # starting position
            record = pickle.load(file)

            if record['name'] == "C":
                record['marks'] += 2

                # move to starting position of the record
                file.seek(pos)
                pickle.dump(record, file) # overwrite data

                # exit loop when the record is updated
                break

        except EOFError:
            break # exit loop if file ends
```

This will add 2 marks to student C's score.

There is a second way to do this as well - write new content to another file, delete original file, rename new file to old file:

```
import pickle
import os

with open("students.dat", "rb+") as file:
    with open("students2.dat", "wb") as file2: # open second file

        while True:
            try:
                record = pickle.load(file)
                if record['name'] == "C":
                    record['marks'] += 2

                pickle.dump(record, file2)

            except EOFError:
                break # exit loop at end of file

# make sure to change file properties once the files have been closed i.e. outside
with

```

```
# remove original file
os.remove("students.dat")

# rename new file
os.rename("students2.dat", "students.dat")
```

The second method is more reliable.

CSV Files

Write to File

Writer object

In order to write to a CSV in the required format, we need to create a writer object:

```
import csv
with open("file.csv", "w", newline="") as file: # opening in write mode
    writer = csv.writer(file)
```

Thus, `writer` is assigned a `csv.writer` object with which we will be able to write to a CSV using the two methods discussed below.

Notice that we added a value for the `newline` parameter. This was done because different operating systems (Windows, MacOS, etc.) have different end of line (EOL) sequences. By removing the default escape sequence in our code, we make it accessible on all devices and prevent a blank line between records. Try removing this in your code and see how the output changes.

Writer Methods

1. `.writerow()`
    - Used to write **one** record (row) to a CSV.
    - It takes a sequence as an input and separates each element in the sequence by the delimiter (by default a comma) in the file

    Example

    ```
    import csv
    with open("file.csv", "w", newline="") as file:
        writer = csv.writer(file)

        data = ["Col1", "Col2", "Col3"]

        writer.writerow(data)
    ```

    When we execute this program, we generate a csv file, `file.csv` with the following content:

    ```
    Col1,Col2,Col3
    ```

2. `.writerows()`
    - Used to write multiple records to a CSV
    - Takes a 2-dimensional nested sequence (sequence within a sequence)
    - Each sequence within the parent sequence represents a record
    - Fields in each record are separated by the delimiter when written to the file

    Example

```
import csv
with open("file.csv", "w", newline="") as file:
    writer = csv.writer(file)

    data = [["Col1", "Col2", "Col3"], ["1", "2", "3"], ["4", "5", "6"]]

    writer.writerows(data)
```

When we run the program we get a file with contents as show below:

```
Col1,Col2,Col3
1,2,3
4,5,6
```

Delimiter

The delimiter is simply the character that fields in a record are separated by. By default, the delimiter for CSV files is a comma, but we can also change it to tab, "|", etc.

Example

```
import csv
with open("file.csv", "w", newline="") as file:
    writer = csv.writer(file, delimiter="\t") # \t represents tab space

    data = [["Col1", "Col2", "Col3"], ["1", "2", "3"], ["4", "5", "6"]]

    writer.writerows(data)
```

When we run the program we get a file with contents as show below:

```
Col1    Col2    Col3
1   2   3
4   5   6
```

As you can see, each fin a record has been separated by a tab space. Thus, the `delimiter` parameter for the `csv.writer` object is the character which separates field in a record.

Quotes

Quoting is putting characters before and after each field in a record, similar to quotes ' '. We can specify the quote character as well as kinds of field to quote.

`quotechar`

This parameter decides the character that acts as a quote

Example

```
import csv
with open("file.csv", "w", newline="") as file:
    writer = csv.writer(file, quotechar="|", quoting=csv.QUOTE_ALL) # don't worry about quoting yet

    data = [["Col1", "Col2", "Col3"], ["1", "2", "3"], ["4", "5", "6"]]

    writer.writerows(data)
```

When we run the program we get a file with contents as show below:

```
|Col1|,|Col2|,|Col3|
|1|,|2|,|3|
|4|,|5|,|6|
```

`quoting`

This parameter decides which kinds of data to quote i.e. it tells the program to quote only elements in a record of a certain kind. To specify which kinds of elements to quote, there are predefined constraints:

1. `csv.QUOTE_ALL` : To quote each and every field in the data
2. `csv.QUOTE_MINIMAL` : To quote fields containing the **delimiter**, **quotechar** or EOL character (eg \n) (see previous pages)
3. `csv.QUOTE_NONNUMERIC` : To quote all text data and convert all numeric data to float
4. `csv.QUOTE_NONE` : To not quote fields (default)

Read File

Reader Object

Similar to the `writer` object, we create a `reader` object to read a CSV

```
import csv
with open("file.csv", "r", newline="") as file: # opening in read mode
    reader = csv.reader(file)
```

The great thing is that we do not need to use any methods to work with the reader object since `csv.reader()` returns a list of lists where each nested list represents a record in the CSV and each element within this list represents the value of a particular field/attribute in that record.

You can think of it as the reverse of `writerows()` since it writes a nested list to the file, whereas `reader()` returns a nested list.

Therefore, assuming we have added the following contents to the file `file.txt`:

```
Col1,Col2,Col3
1,2,3
4,5,6
```

we can print the data record by record:

```
import csv
with open("file.csv", "r", newline="") as file:
    reader = csv.reader(file)

    for record in reader:
        print(record)
```

The output will be:

```
['Col1', 'Col2', 'Col3']
['1', '2', '3']
['4', '5', '6']
```

If we make a modification, so that each individual item is separated by commas:

```
import csv
with open("file.csv", "r", newline="") as file:
    reader = csv.reader(file)

    for record in reader:
        print(",".join(record))
```

In this case, the output will be:

```
Col1,Col2,Col3
1,2,3
4,5,6
```

Solved Examples

1. uniquely identifies the file type

a) Size
b) Format
c) Extension
d) Name

Ans: c) Extension

The file extension such as `.py` or `.txt` is unique to each file type and can be used to easily identify it.

2. `f = open("file.txt", "w")`

What access mode has been used in the above line?

a) Read
b) Binary
c) Text
d) Write

Ans: d) Write

"w" stands for write mode

3. Which of the following is not a function / method of csv module in Python?

(SQP 2021)

a) read()
b) reader()
c) writer()
d) writerow()

Ans: a) read()

read() is a built-in function

4. Which of the following statement opens a binary file `record.bin` **in write mode and writes data from a list** `lst1 = [1,2,3,4]` **on the binary file?**

(SQP 2021)

a)

```
with open('record.bin','wb') as myfile:
    pickle.dump(lst1,myfile)
```

b)

```
with open('record.bin','wb') as myfile:
    pickle.dump(myfile,lst1)
```

c)

```
with open('record.bin','wb+') as myfile:
    pickle.dump(myfile,lst1)
```

d)

```
with open('record.bin','ab') as myfile:
    pickle.dump(myfile,lst1)
```

Ans: a)

The correct syntax to dump is `pickle.dump(object, file)`

5. Which of the following options can be used to read the first line of a text file Myfile.txt?

(SQP 2021)

a) `myfile = open('Myfile.txt'); myfile.read()`
b) `myfile = open('Myfile.txt','r'); myfile.read(n)`
c) `myfile = open('Myfile.txt'); myfile.readline()`
d) `myfile = open('Myfile.txt'); myfile.readlines()`

Ans: c)

`readline()` reads the first line of the text file.

6. A text file student.txt is stored in the storage device. Identify the correct option out of the following options to open the file in read mode.

i. myfile = open('student.txt','rb')
ii. myfile = open('student.txt','w')
iii. myfile = open('student.txt','r')
iv. myfile = open('student.txt')

(SQP 2021)

a) only i
b) both i and iv
c) both iii and iv
d) both i and iii

Ans: c)

By default, the access mode will be `'r'` so iii. and iv. are correct.

7. Which of the following statement is incorrect in the context of binary files?

(SQP 2021)

a. Information is stored in the same format in which the information is held in memory.
b. No character translation takes place
c. Every line ends with a new line character
d. pickle module is used for reading and writing

Ans: c)

Binary files do not have new line characters.

8. Which of the following statement is true?

(SQP 2021)

a. pickling creates an object from a sequence of bytes
b. pickling is used for object serialization
c. pickling is used for object deserialization
d. pickling is used to manage all types of files in Python

Ans: b)

Pickling is another term used for serialization.

9. Syntax of seek function in Python is `myfile.seek(offset, reference_point)` where `myfile` is the file object. What is the default value of `reference_point`?

(SQP 2021)

a) 0
b) 1
c) 2
d) 3

Ans: a)

The default value of the reference point is 0.

10. Syntax for opening Student.csv file in write mode is `myfile = open("Student.csv","w",newline='')`.
What is the importance of newline=''?

(SQP 2021)

a) A newline gets added to the file
b) Empty string gets appended to the first line.
c) Empty string gets appended to all lines.
d) EOL translation is suppressed

Ans: d) EOL translation is suppressed

We use `newline=''` in order to set the newline character to an empty string. If we don't do so, we get a blank line in between all lines.

11. Suppose content of 'Myfile.txt' is

```
Humpty Dumpty sat on a wall
Humpty Dumpty had a great fall
All the king's horses and all the king's men
Couldn't put Humpty together again
```

What will be the output of the following code?

```
myfile = open("Myfile.txt")
record = myfile.read().split()
print(len(record))
myfile.close()
```

(SQP 2021)

a) 24
b) 25
c) 26
d) 27

Ans: c) 26

If we look at the program, we see it is counting the number of words in the file. The number of words is `26`.

12. Suppose content of 'Myfile.txt' is

```
Honesty is the best policy.
```

What will be the output of the following code?

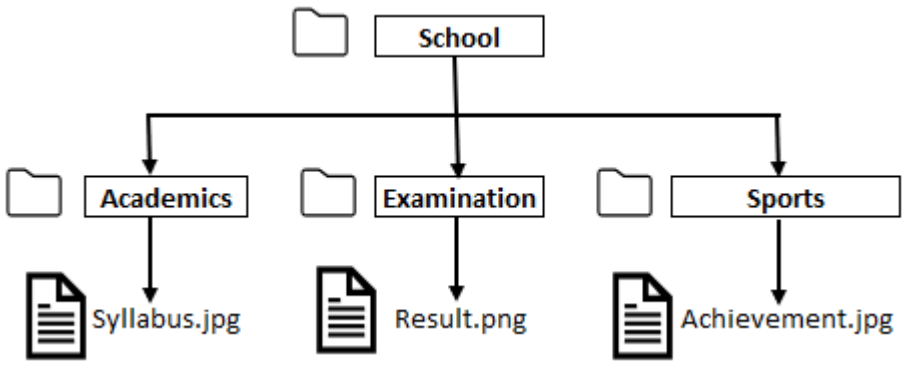
```
myfile = open("Myfile.txt")
x = myfile.read()
print(len(x))
myfile.close()
```

a) 5
b) 25
c) 26
d) 27

Ans: d) 27

We know that the `read` function will return a string of the file contents. In this case it will be a single line containing `27` characters.

13. Consider the following directory structure.



Suppose root directory (School) and present working directory are the same. What will be the absolute path of the file Syllabus.jpg?

(SQP 2021)

a) School/syllabus.jpg
b) School/Academics/syllabus.jpg
c) School/Academics/../syllabus.jpg
d) School/Examination/syllabus.jpg

Ans: b) School/Academics/syllabus.jpg

We know the file is in the `Academics` folder so we can eliminate a and d.

In c, the `..` will take us to the parent directory of `Academics` i.e. `School`, so we can eliminate that too, leaving us with b which is the correct option.

14. Suppose the content of `myfile.txt` is

```
Ek Bharat Shreshtha Bharat
```

What will be the output of the following code?

```
myfile = open("Myfile.txt")
vlist = list("aeiouAEIOU")
vc=0
x = myfile.read()
for y in x:
    if(y in vlist):
    vc+=1
print(vc)
myfile.close()
```

(SQP 2021)

a) 6
b) 7
c) 8
d) 9

Ans: b) 7

Look at the code. You will see it is a code to count the number of vowels in the file.

The vowels are E,a,a,e,a,a,a so a total of 7.

15. Suppose content of 'Myfile.txt' is

```
Twinkle twinkle little star
How I wonder what you are
Up above the world so high
Like a diamond in the sky
Twinkle twinkle little star
```

What will be the output of the following code?

```
myfile = open("Myfile.txt")
line_count = 0
data = myfile.readlines()
for line in data:
    if line[0] == 'T':
        line_count += 1
print(line_count)
myfile.close()
```

(SQP 2021)

a) 2
b) 3
c) 4
d) 5

Ans: a) 2

This code counts the number of lines in the file starting with T. `.readlines()` returns a list of each line and the conditional in the `for` loop checks the starting letter of each line.

16. Suppose content of 'Myfile.txt' is

```
Culture is the widening of the mind and of the spirit.
```

What will be the output of the following code?

```
myfile = open("Myfile.txt")
x = myfile.read()
y = x.count('the')
print(y)
myfile.close()
```

(SQP 2021)

a) 2
b) 3
c) 4
d) 5

Ans: b) 3

The code is counting the number of times the word "the" occurs in the file.

17. Which of the following option is not correct?

a) if we try to read a text file that does not exist, an error occurs.
b) if we try to read a text file that does not exist, the file gets created.
c) if we try to write on a text file that does not exist, no error occurs.
d) if we try to write on a text file that does not exist, the file gets created.

Ans: b) if we try to read a text file that does not exist, the file gets created.

This would result in an error.

18. Assume that the position of the file pointer is at the beginning of 3rd line in a text file. Which of the following option can be used to read all the remaining lines?

(SQP 2021)

a) myfile.read()
b) myfile.read(n)
c) myfile.readline()
d) myfile.readlines()

Ans: d) myfile.readlines()

`.readlines()` reads all the lines from the position of the pointer.

18. What is the significance of the tell() method?

(SQP 2021)

a) tells the path of file
b) tells the current position of the file pointer within the file
c) tells the end position within the file
d) checks the existence of a file at the desired location

Ans: b) tells the current position of the file pointer within the file

19. Which of the following character acts as default delimiter in a csv file?

(SQP 2021)

a) (colon) :
b) (hyphen) -
c) (comma) ,
d) (vertical line) |

Ans: c) (comma) ,

20. What is the correct expansion of CSV files?

(SQP 2021)

a) Comma Separable Values
b) Comma Separated Values
c) Comma Split Values
d) Comma Separation Values

Ans: b) Comma Separated Values

21. Assume the content of text file, `student.txt` is:

```
Arjun Kumar
Ismail Khan
Joseph B
Hanika Kiran
```

```
myfile = open("Myfile.txt")
data_rec = myfile.readlines()
myfile.close()

```

What will be the data type of data_rec?

(SQP 2021)

a) string
b) list
c) tuple
d) dictionary

22 is trying to write a tuple tup1 = (1,2,3,4,5) on a binary file test.bin. Consider the following code written by him.

```
import pickle
tup1 = (1,2,3,4,5)
myfile = open("test.bin",'wb')
pickle._______ #Statement 1
myfile.close()
```

Identify the missing code in Statement 1.

(SQP 2021)

a) dump(myfile,tup1)
b) dump(tup1, myfile)
c) write(tup1,myfile)
d) load(myfile,tup1)

23. Case Study

Rohit, a student of class 12, is learning CSV File Module in Python. During examination, he has been assigned an incomplete python code (shown below) to create a CSV File 'Student.csv' (content shown below). Help him in completing the code which creates the desired CSV File.

CSV File

S. No.	Name	Class	Section
1	AKSHAY	XII	A
2	ABHISHEK	XII	A
3	ARVIND	XII	A
4	RAVI	XII	A
5	ASHISH	XII	A

Incomplete Code

```
import _____ #Statement-1
fh = open(_____, _____, newline='') #Statement-2
stuwriter = csv._____ #Statement-3
data = [ ]
header = ['ROLL_NO', 'NAME', 'CLASS', 'SECTION']
data.append(header)
for i in range(5):
    roll_no = int(input("Enter Roll Number : "))
    name = input("Enter Name : ")
    Class = input("Enter Class : ")
    section = input("Enter Section : ")
    rec = [ _____ ] #Statement-4
    data.append(_____) #Statement-5
stuwriter. _____ (data) #Statement-6
fh.close()
```

(SQP 2021)

i. Identify the suitable code for blank space in the line marked as Statement-1.

a) csv file
b) CSV
c) csv
d) cvs

Ans: c) csv

We use `csv` module to handle CSV files.

ii. Identify the missing code for blank space in line marked as Statement-2.

a) "Student.csv","wb"
b) "Student.csv","w"
c) "Student.csv","r"
d) "Student.cvs","r"

Ans: b) "Student.csv", "w"

The name of the file is `Student.csv` and he wants to open in write mode to create the file, so `"w"`.

iii. Choose the function name (with argument) that should be used in the blank space of line marked as Statement-3.

a) reader(fh)
b) reader(MyFile)
c) writer(fh)
d) writer(MyFile)

Ans: c) writer(fh)

He wants to use a writer object for the file object `fh`.

iv. Identify the suitable code for blank space in line marked as Statement-4.

a) 'ROLL_NO', 'NAME', 'CLASS', 'SECTION'
b) ROLL_NO, NAME, CLASS, SECTION
c) 'roll_no','name','Class','section'
d) roll_no,name,Class,section

Ans: d)

We make a list of the variables with the different values so that the entered values are the elements of the list. If we use strings, the content in the string will be written to the file and not the entered value.

v. Identify the suitable code for blank space in the line marked as Statement-5.

a) data
b) record
c) rec
d) insert

Ans: c) rec

We want to append the `rec` list to the `data` list, which contains all the records to be added to the file.

vi. Choose the function name that should be used in the blank space of line marked as Statement-6 to create the desired CSV File?

a) dump()
b) load()
c) writerows()
d) writerow()

Ans: c) writerows()

We use the `writerows()` function to write multiple records to a csv file, which here are contained in the `data` list.

Chapter 4 MCQs

1. Which of the following is not a known file type?

a) .mp5
b) .csv
c) .pdf
d) .dat

2. What is the full form of EOL?

a) End Of Line
b) End Of List
c) End of Lines
d) End Of Location

3. What format are images and movies stored in?

a) text
b) CSV
c) binary
d) pdf

4. Which of the following is not a reason to use files?

a) To store data permanently
b) For reusability
c) To avoid repetitive tasks of entering data
d) To create a temporary location for data

5. Which of the following file extensions is not human readable?

a) .csv
b) .txt
c) .bin
d) None of the above

6. How is data in text files stored?

a) Binary
b) String
c) Sequence of characters
d) None of the above

7. Which of the following is not a valid pair?

a) \n - newline
b) \t - tab space
c) , - delimiter
d) None of the above

8. Which of the following can be stored in binary files?

a) Images
b) Movies
c) Audio
d) All of the above

9. Binary files can be read by humans

a) True
b) Partially True
c) False
d) Cannot determine

10. What is the default EOL character in Python?

a) '\n'
b) ''\r'
c) ''
d) '\t'

11. What is a file object?

a) Reference to a file
b) It is a file in the computer
c) A keyword
d) A module

12. file object is also known as

a) File handle
b) File copy
c) File directory
d) File link

13.To force python to write the contents of file buffer on to storage file,.....method may be used.

a) buffer()
b) flush()
c) close()
d) write()

14. Which of the following statements are true?

a) When you open a file for reading, if the file does not exist, an error occurs.
b) When you open a file for writing, if the file does not exist, a new file is created.
c) When you open a file for writing, if the file exists, the existing file content is overwritten with the new content.
d) All of the these

15. Default file access mode is _

a) w
b) r+
c) w+
d) r

16. A file maintains a __ which tells us the current position in the file where writing or reading will take place.

a) line
b) file pointer
c) list
d) order

17. Data remains in buffer and is not pushed onto disk until a __ operation is performed.

a) dump( )
b) close( )
c) load( )
d) open( )

18. Which mode is used to retain its previous data and allowing to add new data?

a) write mode
b) read mode
c) open mode
d) append mode

19. Which keyword is associated with opening files?

a) if
b) for
c) while
d) with

20. How to open a text file in write and read mode?

a) r
b) w+
c) w
d) a+

21. Which of the following are valid ways to write content to a file?

a) write
b) writelines
c) Both a) and b)
d) Neither a) nor b)

22. What is the output of the following code?

File:

```
Hello there this is a text file
this is a new line in the file
```

Code:

```
myobject=open("myfile.txt",'r')
print(myobject.readline(10))
myobject.close()
```

a) 10
b) Hello ther
c) Hello there this is a text file
d) Hello there this is a text file\nthis is a new line in the file

23. Which of the following is not a valid way to read file contents?

a) readlines()
b) read()
c) readline()
d) realfile()

24. Which of the following functions return a string?

a) write()
b) read()
c) readlines()
d) writelines()

25. What is the output of the following code?

```
with open("file.txt","w") as file:
    print(file.write("Hello!"))
```

a) Hello!
b) None
c) Error
d) 6

26. Which of the following functions returns a list?

a) writelines()
b) readlines()
c) readline()
d) read()

27. What is the correct syntax to move the pointer to the start of the file?

a) file.seek(0)
b) file.move_pointer(0)
c) file.shift(0)
d) file.move(0)

28. What is the output of the following code?

```
with open("file.txt") as f:
    print(f.tell())
```

a) 0
b) -1
c) 2
d) 1

29. What is the output of the following code?

```
with open("file.txt") as f:
    data = f.read()
    print(f.read())
```

a) -1
b) ""
c) " "
d) None of the above

30. What is the output of the following code for the given file?

File

```
this is one line
this is second line
this is last line
```

Code

```
with open("file.txt") as f:
    data = f.readlines()
    for line in data:
        print(line.split())
```

a)

```
this is one line
this is second line
this is last line
```

b)

```
[this is one line]
[this is second line]
[this is last line]
```

c)

```
['this', 'is', 'first', 'line']
['this', 'is', 'second', 'line']
['this', 'is', 'third', 'line']
```

d)

```
"This is first line"
"This is second line"
"This is third line"
```

31. What is serialization?

a) The process of transforming data in RAM to a bytestream
b) Numbering data in a file
c) Creating a record
d) Counting the number of binary files available on a computer

32. Identify the incorrect pair

a) csv - module for CSV files
b) pickle - module for binary files
c) text - module for text files
d) os - module for operating system functions

33. Which of the following are associated with `.dat` files?

a) dump
b) pickle
c) load
d) All of the above

34. Which of the following is the fastest file to use?

a) CSV
b) binary
c) text
d) None of the above

35. Which of the following error may arise when using a binary file?

a) EOFError
b) LineError
c) SyntaxErrror
d) None of the above

36. Which mode should we use to append and write to a binary file?

a) a+
b) ab+
c) wb+
d) None of the above

37. What is the output of the following code?

```
import pickle

def search(id):
    try:
        while True:
            data = pickle.load(f)
            if data['id'] == id:
                print(data['name'])
                return
    except:
        pass

search(1)
```

a) Ayush
b) Riya
c) Error
d) Raja

38. What is the output of the following code?

```
import pickle

with open("file.dat", "wb") as f:
    pickle.dump("hello", f)

with open("file.dat", "rb") as f:
    data = pickle.load(f)
    print(data)
```

a) Error
b) hello
c) Hello
d) None of the above

39. Which function should we use to write a single record?

a) writerow
b) writerecord
c) addrecord
d) writerec

40. What is the correct way to add the headers Col1, Col2, Col3 for a CSV file?

a) csv.writerow([Col1, Col2, Col3])
b) csv.writerow(["Col1", "Col2", "Col3"])
c) file_object.writerow([Col1, Col2, Col3])
d) writer_object.writerow(['Col1', 'Col2', 'Col3'])

41. What is the default delimiter in CSV files?

a) comma
b) whitespace
c) empty string
d) period

42. What is the correct syntax to create a reader object?

a) reader = csv.reader(file)
b) reader = csv.read(file)
c) reader = csv.readfile(file)
d) reader = csv.readrows(file)

43. Which attribute tells us if a file has been closed or not?

a) isclosed
b) closed()
c) isclosed()
d) closed

44. Mr. Sharma has a grocery shop and stores orders as files to be used by his software. Which files would be of optimal use for him?

a) Python Dictionaries with Text files.
b) Python Dictionaries with Binary files.
c) Python Lists without Binary files.
d) Python Dictionaries without Binary files

45. A student failed to write to a file after executing the command `file = open("file.txt")`. How can he fix this?

a) Using the load function
b) Opening the file in write mode
c) Opening the file in read mode
d) None of the above

46. A curious student accidentally opened a photo in notepad and played around with the contents. What will happen when she opens the file in the photo viewer?

a) The photo will be displayed
b) The photo will have better quality
c) An error will occur
d) None of the above

47. The following is the absolute path to the file you are using: `C:/users/qprogramming/documents/python/file.txt`. How will you use a relative path to open a file `file2.txt` in the documents directory?

a) file2.txt
b) documents/file2.txt
c) ../../file2.txt
d) ../file2.txt

48. Rohan is keeping track of his sales in a table on paper. What is the best way for him to digitize his records and update it in the future?

a) Binary file
b) Text file
c) CSV file
d) Scanner

49. You are working under a tree in the park when a monkey comes and knocks over your laptop, causing it to power off. You were editing a text file whose contents are autosaved. What will happen to your document?

a) They will get destroyed
b) They will be saved
c) They will exist in the recycling bin
d) All of the above

50. Raju was playing around with Python and tried to open a text file in binary mode. What will happen?

a) An error will occur
b) The system will crash
c) The file will open
d) None of the above

Chapter 4 MCQ Answers

Q	A	Q	A	Q	A	Q	A	Q	A
1	A	11	A	21	C	31	A	41	A
2	A	12	A	22	B	32	C	42	A
3	C	13	B	23	D	33	D	43	D
4	D	14	D	24	B	34	B	44	B
5	C	15	D	25	D	35	A	45	B
6	A	16	B	26	B	36	D	46	C
7	D	17	B	27	A	37	C	47	D
8	D	18	D	28	A	38	B	48	C
9	C	19	D	29	B	39	A	49	B
10	A	20	B	30	C	40	D	50	C

Chapter 4 Output Finding Questions

1. You are given a file `file.txt` with the following contents:

```
ABCDEFGHIJKLMNOP
QRSTUVWXYZ
NOW I KNOW MY ABCs
NEXT TIME WON'T YOU SING WITH ME
```

Find the output of the following code:

```
with open("file.txt"):
    content = file.read()
    total = 0

    for ch in content:
        if ch.islower():
            total+=1
print(total)
```

a) 0
b) 1
c) 2
d) 3

2. You are given a file `file.txt` with the following contents:

```
The truth always prevails.
```

Find the output of the following code:

```
with open("file.txt"):
    content = file.read()
    total = 0

    for ch in content:
        if ch.lower() == "t":
            total+=1
print(total)
```

a) 0
b) 1
c) 2
d) 3

3. You are given a file `file.txt` with the following contents:

```
Silence is golden, speech is silver.
```

Find the output of the following code:

```
with open("file.txt"):
    content = file.read()
    print(content.count("is"))
```

a) 0
b) 1
c) 2
d) 3

4. You are given a file `file.txt` with the following contents:

```
The quick brown
fox jumps
over the
lazy
dog
```

Find the output of the following code:

```
with open("file.txt"):
    content = file.readlines()
    total = len(content)

print(total)
```

a) 3
b) 4
c) 5
d) 6

5. You are given a file `file.txt` with the following contents:

```
The sun
rises in the east
and sets
in the west.
At dawn and dusk the sun
rises and sets
bringing life to the planet
and putting it to sleep
```

Find the output of the following code:

```
with open("file.txt"):
    content = file.read()
    content = content.split()
    total = len(content)

print(total)
```

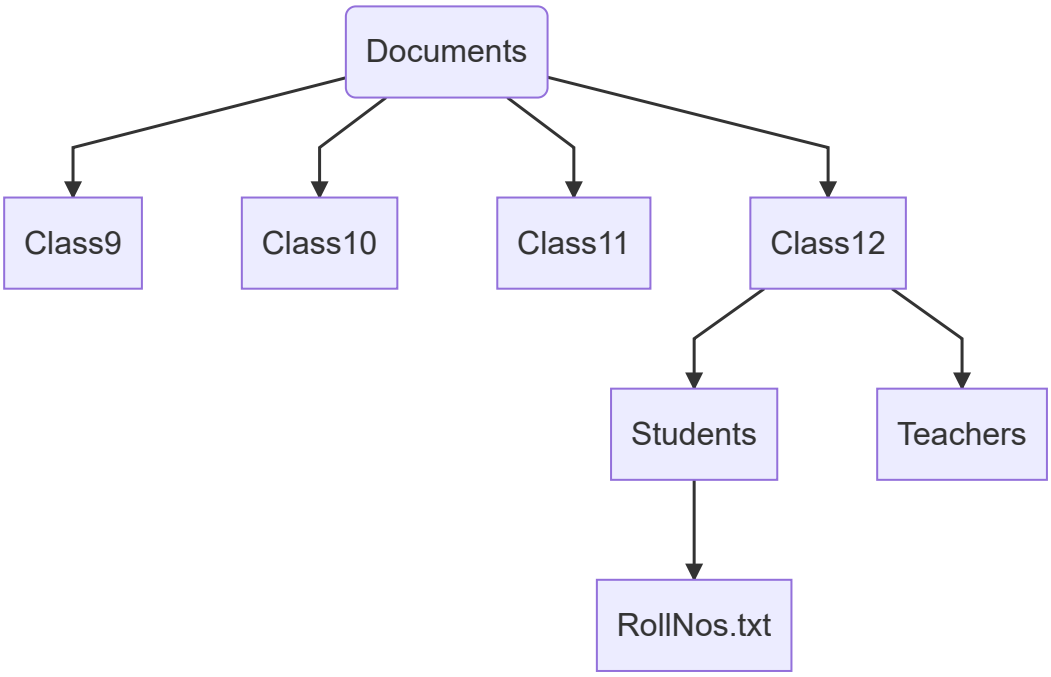
a) 30
b) 31
c) 32
d) 33

Chapter 4 Output Finding Answers

Q	A	Q	A	Q	A	Q	A	Q	A
1	B	2	D	3	C	4	C	5	A

Chapter 4 Diagram Based Questions

All the following questions are based on the following diagram:



Assume that the current working directory is `Documents` for all questions

1. Fill in the code to get to `Class9`

```
import os
os.chdir(_______)
```

a) Class9
b) "Class9"
c) Documents/Class9
d) Any of the above

2. What is the path to `RollNos.txt`?

a) Documents/Class12/Teachers/RollNos.txt
b) Documents/Class11/Students/RollNos.txt
c) /Class12/Students/RollNos.txt
d) /Class11/Students/RollNos.txt

3. Let us say the CWD has changed to `Students`. Fill in the code to go back to `Documents` as the CWD

```
import os
os.chdir(______)
```

a) ./.
b) ./..
c) /..
d) ../..

4. Let's assume the absolute path to `Documents` is `C:/users/qprogramming/Documents`. What is the absolute path of `Teachers`?

a) C:/users/qprogramming/Documents/Teachers
b) C:/users/qprogramming/Documents/Class11/Teachers
c) C:/users/qprogramming/Document/Class12/Teachers
d) C:/users/qprogramming/Documents/Class12/Teachers

Chapter 4 Diagram Questions Answers

Q	A	Q	A	Q	A	Q	A
1	B	2	C	3	D	4	D

Chapter 4 Case Study

1. You are working in a firm and need to create a database for your 10 clients. Complete the code to do so.

```
import _________ # statement 1

with open("database.csv", "w", _________) as f:# statement 2
    writer = csv.writer(f)

    header = _________________ # statement 3
    writer._________(header) # statement 4

    for i in range(____): # statement 5
        name = input("Enter name: ")
        payment = input("Enter amount: ")

        record = __________ # statement 6

        writer.writerow(record)
```

i. Choose the correct option for statement 1

a) CSV
b) VCS
c) csv
d) commasepvalues

ii. Choose the statement to prevent blank lines in the file

a) blankline = False
b) newline = ""
c) blank=False
d) Do not add anything

iii. Choose the best option for statement 3

a) [name, payment]
b) ["name", payment]
c) name, payment
d) ["Name", "Payment"]

iv. Choose the correct function in statement 4

a) write
b) writerow
c) writerecord
d) writeheader

v. Fill in statement 5

a) 100
b) 10
c) 1000
d) 0

vi. Choose the best option to complete statement 6

a) ["name","payment"]
b) [name, "payment"]
c) "name", "payment"
d) [name, payment]

2. You are a programmer for a big data science company. You need to store all the data for easy access. In this scenario, fill in the code to complete your task.

```
import ________ # statement 1

with open("data.dat", _______) as file: #statement 2

    data = [1,2,3,4,5,6]

    pickle._____(data, file) # statement 3
```

i. Select the most appropriate option to fill statement 1

a) pickle
b) csv
c) binary
d) Any of the above

ii. Select the best option for statement 2 to create the file and write to it.

a) "r"
b) "w"
c) "wb"
d) "r+"

iii. Select the correct function in statement 3

a) load
b) dump
c) pickle
d) serialize

iv. Your friend asks why you have not used a text file since it would be easier to use. What do you say?

a) That's a good idea, I will do that
b) Text files cannot store a my data
c) Binary files are much faster to use
d) None of the above

v. What is another term for pickling?

a) Dumping
b) Bytestream
c) Deserialization
d) Serialization

vi. What is the correct way to read the file `data.dat`?

a) f = open("data.dat")
b) f = open("data.dat", "rb")
c) f = open("data.dat", "r")
d) f = open("data.dat", "wb")

3. Your friend has written a short article about the environment in a text file called `article.txt`.

```
Environment refers to the natural world on mars, especially as affected by human activty.
The environment on mars has 2 components, biotic and abiotic.
The environment on mars has been bad since the last decade. Changes need to be made so that there is a better future for mars.
```

To make some changes in it, they made a python program. But they need some help finishing it, so they ask for your assistance

```
with open("article.txt",_______) as file:   #statement 1
    content=file._______                    #statement 2
    new=content._________                   #statement 3
    ________                                #statement 4
    file.write(_______)                     #statement 5
```

i. What mode should the file be opened in Statement 1 so that it is possible to read and write in it, but a new file should *not* be created?

a) w+
b) w
c) r
d) r+

ii. Which command should be used in statement 2 to read the entire contents of the file and store it as a string

a) read()
b) readline()
c) readlines()
d) read(4)

iii. Which inbuilt string method should be used in statement 3 so that the new string contains a copy of the old string where the word 'mars' is changed to 'earth'

a) change('mars','earth')
b) remove('mars','earth')
c) replace('mars','earth')
d) change_to('mars','earth')

iv. To write the new string into the file by removing old content, what do we need to make sure of?

a) We need to first convert it to a list
b) The file pointer is at the beginning of the file after reading, so we must bring it to the end
c) The file pointer is at the end of the file after reading, so we must bring it back to the beginning
d) Nothing, we can write the new string directly

v. What command should be used in statement 4 to implement what is to be done in part iv

a) file.seek(0)
b) No need for any command over there
c) new=list(new)
d) file.seek(2)

vi. Write what should be filled in the blank of statement 5

a) file
b) new
c) content
d) seek

Chapter 4 Case Study Answers

Case	i.	ii.	iii.	iv.	v.	vi.
Case Study 1	C	B	D	B	B	D
Case Study 2	A	C	B	C	D	B
Case study 3	D	A	C	C	A	B