

Python Language

Questions for Revision

Try answering the following questions to test your understanding of the topics covered in this notebook:

1. What is a variable in Python?

Ans. A variable in the Python program stores/provides data to the computer for further processing.

2. How do you create a variable?

Ans. The variables in python can be defined in many ways. Examples are as follows:

Example:

```
x=10

y="Shabbir Hussain"

z=10.125

c1, c2, c3 = "Red", "Green", "Blue"

print(x)

print(y)

print(z)

print(c1)

print(c2)

print(c3)
```

Output:

```
10
Shabbir Hussain
10.125
```

Red
Green
Blue

3. How do you check the value within a variable?

Ans. By printing the value of the variable

4. How do you create multiple variables in a single statement?

Ans. Multiple variables can be defined in the following way:

Color1, Color2, Color3 = "Red", "Green", "Blue"

5. How do you create multiple variables with the same value?

Ans. Var1 = Var2 = Var3 = Var4 = Var5 = "Pakistan Zinda Abad"

6. How do you change the value of a variable?

Ans. Just reassign a new value to a variable. System will consider the new value after defining the new value.

Example:

X=10

Print("Value of X = ",X) ~ The answer will be 10

X=20

Print("Value of X = ",X) ~ The answer will be 20

X=30

Print("Value of X = ",X) ~ The answer will be 30

7. How do you reassign a variable by modifying the previous value?

Ans. Use += operator to modify the value previously stored in the variable. We can also use variable = variable + any_Number

Var=100

Var

The Answer will be 100

Var = Var + 25

Var

The Answer will be 125

Example 2:

Var=100

Var

Var += 25

Var

Output:

100

125

8. What does the statement counter += 4 do?

Ans. It will add 4 to the value already stored in the variable.

Example 1:

Var=100

Var

The Answer will be 100

Var = Var + 4

Var

The Answer will be 104

Example 2:

Var=100

Var

The Answer will be 100

Var += 4

Var

The Answer will be 108

9. What are the rules for naming a variable?

Ans. Keywords can't be taken as variable Name

The Variable name can't be started with a number i.e 1Var, 2Var etc.

Python is a case sensitive language so variable names var, VAR, Var are different from each other

Special character can't be written inside the variable names

10. **Are variable names case-sensitive? Do `a_variable`, `A_Variable`, and `A_VARIABLE` represent the same variable or different ones?**

Ans. Python is a case sensitive language and all above mentioned variable names are different from each other.

11. **What is Syntax? Why is it important?**

Ans. Syntax are the rules of the language and these must be followed while writing commands. If we do not follow syntax it will cause a SYNTAX ERROR.

12. **What happens if you execute a statement with invalid syntax?**

Ans. if we do not follow syntax it will cause a SYNTAX ERROR.

13. **How do you check the data type of a variable?**

Ans. `Type()` function is used to check the data type of a variable

14. **What are the built-in data types in Python?**

Ans. Followings are the builtin data types in Python

Int

Float

String

Boolean

None

List

Tuple

Dictionary

15. **What is a primitive data type?**

Ans. Primitive Types are the most basic data structures. They are the building block for data manipulation, and such contains pure, simple values of data. Python has four primitive types: **integers, floats, booleans and strings**.

16. **What are the primitive data types available in Python?**

Ans. Python has four primitive types: **integers, floats, booleans and strings**.

17. What is a data structure or container data type?

Ans. Containers are any object that holds an arbitrary number of other objects. Generally, containers provide a way to access the contained objects and to iterate over them. Examples of containers include tuple , list , set , dictionary etc.

18. What are the container types available in Python?

Ans. The Python builtin container types are tuple, list, dict, set, frozenset and str and unicode (or bytes and str in Python 3), as well as a couple other constructs that are technically types but are not commonly used outside of specific contexts (e.g., buffer objects and xrange objects).

19. What kind of data does the Integer data type represent?

Ans. Non Decimal data can only be represented by integers.

20. What are the numerical limits of the integer data type?

Ans. There is no limit for integers to store integer data. There were two types of integer in the old Python i.e int and long where long was sporting a bigger number than integer.

21. What kind of data does the float data type represent?

Ans. Float data types can represent data with and without decimal place.

22. How does Python decide if a given number is a float or an integer?

Ans. Python evaluates a number with decimal, if a number has a decimal place then it will be considered as float otherwise integer. For example x=1.0 will be considered as Float even if it has ZERO after decimal which has no meaning but it has one decimal place. On the other hand x=1 will be considered as an integer as it has no decimal place.

23. How can you create a variable which stores a whole number, e.g., 4 but has the float data type?

Ans. We can define a variable to store a whole number 4 like this (x=4.0).

24. How do you create floats representing very large (e.g., 6.023×10^{23}) or very small numbers (0.000000123)?

Ans. By using the following expression we can define these variables.

Example:

```
# Exponent Power Expression in Python
```

```
import math
```

```
small = 0.000000123
```

```
exponent = 23
```

```
large = 6.023 * (10**exponent)
```

```
print(small)
```

```
print(large)
```

Output:

```
1.23e-07
```

```
6.022999999999999e+23
```

25. What does the expression 23e-12 represent?

Ans. The answer of the expression will be **2.3e-11**

26. Can floats be used to store numbers with unlimited precision?

Ans. No

27. What are the differences between integers and floats?

Ans. Integers can't store data with precision but Floats can store data with and without precision.

28. How do you convert an integer to a float?

Ans. We can convert an integer to float by using the Float() function.

29. How do you convert a float to an integer?

Ans. We can convert a float to an integer by using the int() function.

30. What is the result obtained when you convert 1.99 to an integer?

Ans. By converting a float number 1.99 to an integer the answer will be without decimal place i.e 1. The decimal place will be lost by converting a float to an integer.

31. What are the data types of the results of the division operators / and //?

Ans. The data type of the division operation “/” will be float and the data type of “//” will be Integer.

```
# using single / operator

x = 10

y = 3

z = x/y

print("Using Single / operator ",z)

# using double // operator

z = x//y

print("Using Single // operator ",z)
```

32. What kind of data does the Boolean data type represent?

Ans. TRUE or FALSE

33. Which types of Python operators return booleans as a result?

Ans. Logical Operators (Equal, Non Equal, Greater than, Lesser Than etc...) will produce a boolean output like True or False.

34. What happens if you try to use a boolean in an arithmetic operation?

Ans. True will be considered as 1 (ONE) and False will be considered as 0 (ZERO)

35. How can any value in Python be covered to a boolean?

The value False itself

Ans: Following are the covered boolean values in different data types. These values are considered as FALSE in boolean data type.

Integer 0

Float 0.0

Empty value None

Empty text ""

Empty list []

Empty tuple ()

Empty dictionary {}

Empty set set()

Empty range range(0)

36. What are truthy and falsy values?

Ans. The following values will be considered as FALSE and other than these values will be considered as TRUE.

Integer 0

Float 0.0

Empty value None

Empty text ""

Empty list []

Empty tuple ()

Empty dictionary {}

Empty set set()

Empty range range(0)

37. What are the values in Python that evaluate to False?

Ans.

Integer 0

Float 0.0

Empty value None

Empty text ""

Empty list []

Empty tuple ()

Empty dictionary {}

Empty set set()
Empty range range(0)

38. Give some examples of values that evaluate to True.

Ans.

Integer 1
Float 1.0
Empty text "Shabbir Hussain"
Empty list [1 1 0 0]
Empty tuple (10)

39. What kind of data does the None data type represent?

Ans. The None keyword is used to define a null variable or an object. In Python, the None keyword is an object, and it is a data type of the class NoneType . We can assign None to any variable, but you can not create other NoneType objects. Note: All variables that are assigned None point to the same object.

40. What is the purpose of None?

Ans. It is the same as Void in C and c++ languages. The None keyword is used to define a null value, or no value at all. None is not the same as 0, False, or an empty string.

41. What kind of data does the String data type represent?

Ans. String data type represents **TEXT** data.

42. What are the different ways of creating strings in Python?

Ans. We can define a string either by using **Double or Single** quote. We can also use wildcards "\n" for new lines and "\t" for Tab.

```
today = "Saturday"
```

```
today = 'Saturday'
```

```
week="Followings are the weekdays\nSaturday \nSunday \nMonday \nTuesday\nWednesday \nThursday \nFriday"
```

```
print(week)
```

Note: If we will not use print function then the wildcard will not work. The

answer of this code will be as follows:

Followings are the weekdays

Saturday

Sunday

Monday

Tuesday

Wednesday

Thursday

Friday

Program Code:

```
# using double quotes

today = "Saturday"

# using single quotes

tomorrow = 'Sunday'

# using wildcards

week="Followings are the weekdays\nSaturday \nSunday \nMonday
\nTuesday \nWednesday \nThursday \nFriday"

print(today)

print(tomorrow)

print(week)
```

Output:

Saturday

Sunday

Followings are the weekdays

Saturday

Sunday

Monday

Tuesday

Wednesday

Thursday

Friday

We can also use the “\t” wildcard to add a TAB between words. **For example:**

```
week="Followings are the weekdays\nSaturday \tSunday \tMonday \tTuesday\n\tWednesday \tThursday \tFriday"
```

```
print(week)
```

Followings are the weekdays

Saturday Sunday Monday Tuesday Wednesday Thursday
Friday

For example If we will use only variable name to print data then the output will be as follows:

```
week
```

```
Followings are the weekdays\nSaturday \nSunday \nMonday \nTuesday\n\nWednesday \nThursday \nFriday
```

43. **What is the difference between strings created using single quotes, i.e. ' and ' vs. those created using double quotes, i.e. " and "?**

Ans. In Python, such sequences of characters are included inside single or double quotes. As far as language syntax is concerned, there is no difference in single or double quoted strings. Both representations can be used interchangeably.

44. **How do you create multi-line strings in Python?**

Ans. Use single or double quote three times at the beginning and end of the multiline string (" " ") or (' ' ')

Example:

```
multi_Line_String = " " " My Name is Shabbir Hussain
```

```
My Father name is Abdul Aziz Bhatti
```

```
I lived in Kasur " " "
```

```
print(multi_Line_String)
```

The output will be

My Name is Shabbir Hussain

My Father name is Abdul Aziz Bhatti

I lived in Kasur

45. What is the newline character, \n?

Ans. The “\n” is a new line character. It is a non printing character and it will print its behaviour instead of physical character. I.e the system will insert a new line in the string wherever we will use it inside the string.

Note: Keep in mind we must use print() function to print the variable having \n otherwise \n will be printed as it is without inserting a new line.

Example:

```
week="Followings are the weekdays\nSaturday \nSunday \nMonday \nTuesday\nWednesday \nThursday \nFriday"
print(week)
```

Output will be as follows:

```
Followings are the weekdays
Saturday
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
```

46. What are escaped characters? How are they useful?

Ans. In Python strings, the backslash "\" is a special character, also called the "escape" character. It is used in representing certain whitespace characters: "\t" is a tab, "\n" is a newline, and "\r" is a carriage return.

1. \n is used for new line character
2. \t is used to insert a TAB
3. \r is used to insert Carriage Return

47. How do you check the length of a string?

Ans. len() function is used to check the length of the string.

48. How do you convert a string into a list of characters?

Ans. We can convert a string into a list of characters by using list() function

Example:

```
input = 'ABC'
chars = list(input)
print(chars)
```

Output:

```
['A', 'B', 'C']
```

49. How do you access a specific character from a string?

Ans. Following example shows how to access a specific character from a string.

Example:

```
input = 'ABC'
chars = list(input)
print(chars[1])
```

Output:

```
B
```

50. How do you access a range of characters from a string?

Ans. To access the range of characters, a special character ":" is used. The syntax is as follows:

```
Var_Name[Start : End]
```

Example:

```
input = 'ABCDEF'
chars = list(input)
print(chars)
print(chars[0:3])
```

Output:

```
['A', 'B', 'C', 'D', 'E', 'F']
['A', 'B', 'C']
```

Note: The string starts from index 0 which means in the above mentioned string "A" will be at 0 index, "B" will be at 1 index and so on.

```
0 1 2 3 4 5  
['A', 'B', 'C', 'D', 'E', 'F']
```

51. How do you check if a specific character occurs in a string?

Ans. A specific character/word can be searched using the **in** operator. It returns **True or False**

Example:

```
my_Country = "Pakistan"  
'P' in my_Country
```

Output:

True

52. How do you check if a smaller string occurs within a bigger string?

Ans. A specific/smaller string/characters/word can be searched using the **in** operator. It returns **True or False**

Example:

```
my_Country = "Pakistan"  
'Pak' in my_Country
```

Output:

True

53. How do you join two or more strings?

Ans. Plus "+" operator is used to join two or more than two strings. Example is as follows:

Example:

```
first_Name = "Shabbir"  
last_Name = "Hussain"  
family_Name = "Bhatti"  
full_Name = first_Name + last_Name + family_Name  
  
print("My name is ",full_Name)
```

Output:

My name is Shabbir Hussain Bhatti

54. What are "methods" in Python? How are they different from functions?

Ans. The Python method is called on an object, unlike a function. We call a method on an object, it can access the data within it. A method may alter an object's state, but Python function usually only operates on it, and then prints something or returns a value.

55. What do the .lower, .upper and .capitalize methods on strings do?

Ans. These are text functions used to convert the text into lower case letters, upper case letters and proper case letters. The example is as follows:

```
first_Name = "Shabbir"
last_Name = "Hussain"
family_Name = "Bhatti"

lower_Case = first_Name.lower()
upper_Case = last_Name.upper()
capital_Case = family_name.capitalize()

lower_Case
upper_Case
capital_Case
```

Output:

```
shabbir
HUSSAIN
Bhatti
```

56. How do you replace a specific part of a string with something else?

Ans. A .replace() function is used to replace a text in the string. The example is as follows:

Example:

```
today_Day = "Sunday"
tomorrow = today_Day.replace("Sun", "Mon")
```

```
print("Today is ",today_Day)
print("Tomorrow will be ",tomorrow)
```

Output:

```
Today is Sunday
Tomorrow will be Monday
```

57. How do you split the string "Sun,Mon,Tue,Wed,Thu,Fri,Sat" into a list of days?

Ans. A string can be splitted by using the .split() function in python. The example is as follows:

Example:

```
week_Days = "Sun,Mon,Tue,Wed,Thu,Fri,Sat"
split_Days = week_Days.split(",")
print(split_Days)
```

Output:

```
['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']
```

58. How do you remove whitespace from the beginning and end of a string?

Ans. There are three different functions used for the similar purpose these are as follows:

variable_Name.lstrip() => used to remove whitespaces from left side of the str

variable_Name.rstrip() => used to remove whitespaces from right side of the str

variable_Name.strip() => is used to remove whitespaces from both side of the str

Examples:

```
name = '  Shabbir Hussain  '
```

```
#remove spaces from left
```

```
print (name.lstrip())
```

```
#remove spaces from right
```

```
print (name.rstrip())
```

```
#remove spaces from both side
```

```
print (name.strip())
```


Output:

Shabbir Hussain.....

.....Shabbir Hussain

Shabbir Hussain

59. What is the string format method used for? Can you give an example?

Ans. The `format()` method formats the specified value(s) and inserts them inside the string's placeholder.

The placeholder is defined using curly brackets: `{}`. Read more about the placeholders in the Placeholder section below.

The `format()` method returns the formatted string.

Syntax is as follows:

string.format(value1, value2...)

Examples are as follows:

```
myself1 = "My name is {fname}, I'm {age}".format(fname = "Shabbir", age = 42)
```

```
myself2 = "My name is {0}, I'm {1}".format("Shabbir",42)
```

```
myself3 = "My name is {}, I'm {}".format("Shabbir",42)
```

```
My name is Shabbir, I'm 42
```

```
My name is Shabbir, I'm 42
```

```
My name is Shabbir, I'm 42
```

60. What are the benefits of using the .format method instead of string concatenation?

Ans. Format Function in Python (`str.format()`) is the technique of the string category that permits you to try and do variable substitutions and data formatting. It enables you to concatenate parts of a string at desired intervals through point data format.

61. How do you convert a value of another type to a string?

Ans. We can use str data type to convert a value of any data type (int, float, boolean etc) into a string data type. Examples are as follows:

```
str(125)
```

```
str(125.125)
```

```
str(True)
```

```
Str(False)
```

62. How do you check if two strings have the same value?

Ans. Double Equal “==” operator is used to check whether both variables have the same value or not. It will return True or False value.

```
first_name = “Shabbir”
```

```
last_name = “Bhatti”
```

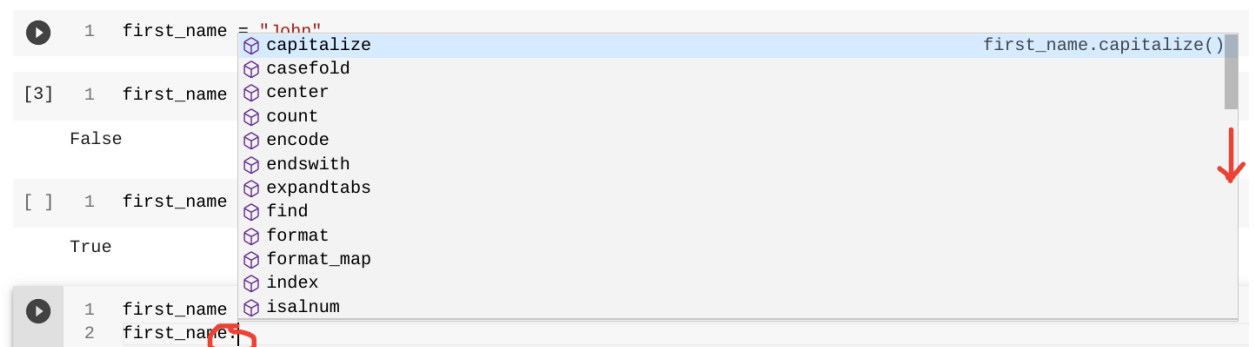
```
first_name == last_name
```

Output:

False

63. Where can you find the list of all the methods supported by strings?

Ans. We can access all the methods available and supported by string by using DOT (.) after the name of the string variable. The example is as follows:



64. What is a list in Python?

Ans. Lists are used to store multiple items in a single variable. Lists are one of the built-in data types in Python. The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets.

65. How do you create a list?

Ans. In Python programming, a list is created by placing all the items (elements) inside square brackets [], separated by commas. It can have any number of items and they may be of different types (integer, float, string etc.). List can be created in the following ways:

```
# empty list
```

```
my_list = []
```

```
# list of integers
```

```
my_list = [1, 2, 3]
```

```
# list with mixed data types
```

```
my_list = [1, "Hello", 3.4]
```

A list can have another list as one element inside the list.

```
# nested list
```

```
my_list = ["mouse", [8, 4, 6], ['a']]
```

```
new_list = ["one", "two", my_list, True, 125, 5.24]
```

66. Can a Python list contain values of different data types?

Ans. Yes a python list can have values of different types even if it may have another list.

```
# list with different data types as well as another list as an element.
```

```
my_list = ["mouse", [8, 4, 6], ['a']]
```

```
new_list = ["one", "two", my_list, True, 125, 5.24]
```

67. Can a list contain another list as an element within it?

Ans. Yes, a list may have another list as an element inside the main list..

```
# list with another list as an element.
```

```
my_list = ["mouse", [8, 4, 6], ['a']]
```

```
new_list = ["one", "two", my_list, True, 125, 5.24]
```

68. Can you create a list without any values?

Ans. Yes an empty list can be defined without having any element.

```
# empty list  
my_list = []
```

69. How do you check the length of a list in Python?

Ans. len() function is used to check the length of a list in Python language.

```
week_Days = ["Saturday", "Sunday", "Monday", "Tuesday", "Wednesday",  
"Thursday", "Friday"]  
print("Days in a week are ",len(week_Days))
```

Output:

Days in a week are 7

70. How do you retrieve a value from a list?

Ans. All the elements are stored inside the list on a specific index number starting from 0 and so on. We can access a specific element by using its index number within the parentheses of the list data type variable name.

Example:

```
week_Days = ["Saturday", "Sunday", "Monday", "Tuesday", "Wednesday",  
"Thursday", "Friday"]  
print("Today is ",week_Days[0])
```

Output:

Today is Saturday.

71. What is the smallest and largest index you can use to access elements from a list containing five elements?

Ans. The index is starting from ZERO and the last number will be one less from the total number of elements. In this question, there are five elements so

Smallest Element Index Number = 0

Largest Element Index Number = 4 ~ (5-1)

72. What happens if you try to access an index equal to or larger than the size of a list?

Ans. A syntax error will happen in case we will use the index number equal to or larger than the size of the list. The will look like as follows:

```
week_Days = ["Saturday", "Sunday", "Monday", "Tuesday", "Wednesday",  
"Thursday", "Friday"]  
print(week_Days[10])
```

Error:

IndexError Traceback (most recent call last)

[<ipython-input-14-b8c91da6ba3a>](#) in <module>()

----> 1 week_Days[10]

IndexError: list index out of range

73. What happens if you try to access a negative index within a list?

Ans. If we use a negative index number it will start from the last index of the list. Example is as follows:

```
week_Days = ["Saturday", "Sunday", "Monday", "Tuesday", "Wednesday",  
"Thursday", "Friday"]  
print(week_Days[-1])  
print(week_Days[-2])  
print(week_Days[-3])
```

Output:

```
Friday  
Thursday  
Wednesday
```

74. How do you access a range of elements from a list?

Ans. A colon (:) is used to access a range of elements from a list.

Example:

```
week_Days = ["Saturday", "Sunday", "Monday", "Tuesday", "Wednesday",  
"Thursday", "Friday"]  
print(week_Days[0:3])
```

Output:

```
['Saturday', 'Sunday', 'Monday']
```

- 75. How many elements does the list returned by the expression `a_list[2:5]` contain?**

Ans. Three elements will be returned because the range 2:5 includes the elements starting from index 2 but does not include the element at the end index 5. So index 2, 3 and 4 will be returned from this expression.

- 76. What do the ranges `a_list[:2]` and `a_list[2:]` represent?**

Ans. The range `a_list[:2]` will return the first two elements which means this expression starts from index ZERO and ends before index 2.

On the other hand the expression `a_list[2:]` will return all the elements starting from index 2.

- 77. How do you change the item stored at a specific index within a list?**

Ans. The value of the variable on a specific index can be changed at any point of time by using a simple equal to (=) operator with a new value. The example is as follows:

Example:

```
week_Days = ["Saturday", "Sunday", "Monday", "Tuesday", "Wednesday",  
"Thursday", "Friday"]  
week_Days[1] = "Holiday"  
print(week_Days[0:3])
```

Output:

```
['Saturday', 'Holiday', 'Monday']
```

****Another Example:****

```
prices = [149.95, 72.50, 30.00, 29.95, 55.00]

print(prices)

prices[0] = 104.95

print(prices)

prices[0] = prices[0] + 5

print(prices)
```

****Output:****

```
[149.95, 72.50, 30.0, 29.95, 55.0]

[104.95, 72.50, 30.0, 29.95, 55.0]

[109.95, 72.50, 30.0, 29.95, 55.0]
```

78. How do you insert a new item at the beginning, middle, or end of a list?

Ans. A special list function `.insert()` is used to insert an item on a specific location either beginning, middle, end or anywhere in the list by mentioning index number and new item within parentheses. The syntax is as follows:

`list_Name.insert(index, value)`

Example:

```
prices = [149.95, 72.50, 30.00, 29.95, 55.00]

print(prices)

# changing value in the list

prices[0] = 104.95

print(prices)

# updating value of an item in the list

prices[0] = prices[0] + 5
```

```
print(prices)

# inserting an item in the list

prices.insert(1,125.0)

print(prices)
```

Output:

```
[149.95, 72.5, 30.0, 29.95, 55.0]

[104.95, 72.5, 30.0, 29.95, 55.0]

[109.95, 72.5, 30.0, 29.95, 55.0]

[109.95, 125.0, 72.5, 30.0, 29.95, 55.0]
```

79. How do you remove an item from a list?

Ans. We can delete one or more items from a list using the keyword **del**. It can even delete the list entirely.

Example:

```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']
```

```
# delete one item
```

```
del my_list[2]
```

```
print(my_list)
```

```
# delete multiple items
```

```
del my_list[1:5]
```

```
print(my_list)
```

```
# delete entire list
```

```
del my_list
```

```
# Error: List not defined
```

```
print(my_list)
```


Output:

```
['p', 'r', 'b', 'l', 'e', 'm']
```

```
['p', 'm']
```

Traceback (most recent call last):

File "<string>", line 18, in <module>

NameError: name 'my_list' is not defined

We can also use `remove()` method to remove the given item or `pop()` method to remove an item at the given index.

The `pop()` method removes and returns the last item if the index is not provided. This helps us implement lists as stacks (first in, last out data structure).

We can also use the `clear()` method to empty a list.

Example:

```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# print complete list

print(my_list)

# removign p from the list using .remove() function

my_list.remove('p')

# print list after removing first p

print(my_list)

# removing first element using pop() function

print(my_list.pop(1))

# print list after .pop() function

print(my_list)
```

```
# clear all items in the list

my_list.clear()

# print after calling .clear() function

print(my_list)
```

Output:

```
['p', 'r', 'o', 'b', 'l', 'e', 'm']

['r', 'o', 'b', 'l', 'e', 'm']

o

['r', 'b', 'l', 'e', 'm']

m

['r', 'b', 'l', 'e']

[]
```

Finally, we can also delete items in a list by assigning an empty list to a slice of elements.

```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

print(my_list)

# items from index 2 will be empty

my_list[2:3] = []

print(my_list)

# items from index 2 to index 4 will be empty

my_list[2:5] = []

print(my_list)
```

Output:

```
['p', 'r', 'o', 'b', 'l', 'e', 'm']  
  
['p', 'r', 'b', 'l', 'e', 'm']  
  
['p', 'r', 'm']
```

80. How do you remove the item at a given index from a list?

Ans. We can delete an item at a given index by using del function and mentioning the index in the prosthesis after the variable name.

Example:

```
# delete one item  
my_list = ['p','r','o','b','l','e','m']  
del my_list[2]  
print(my_list)
```

Output:

```
['p', 'r', 'b', 'l', 'e', 'm']
```

81. How do you check if a list contains a value?

Ans. A specific value can be searched using the **in** operator. It returns **True** or **False**

82. How do you combine two or most lists to create a larger list?

Ans. Keep two or more lists inside the large list as an element to create a bigger list. The example is as follows:

Example:

```
Q01 = ["Jan", "Feb", "Mar"]  
Q02 = ["Apr", "May", "Jun"]  
Q03 = ["Jul", "Aug", "Sep"]  
Q04 = ["Oct", "Nov", "Dec"]  
financial_Year = Q01 + Q02 + Q03 + Q04  
print(financial_Year)
```

Output:

```
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',  
'Oct', 'Nov', 'Dec']
```

83. How do you create a copy of a list?

Ans. There are many ways to copy an old list to a new list. Some of them are as follows:

1. You can use the builtin list.copy() method (available since Python 3.3):
new_list = old_list.copy()
2. You can slice it: new_list = old_list[:]
3. You can use the built in list() function: new_list = list(old_list)

The example is as follows:

Example:

```
Q01 = ["Jan", "Feb", "Mar"]  
Q02 = ["Apr", "May", "Jun"]  
Q03 = ["Jul", "Aug", "Sep"]  
Q04 = ["Oct", "Nov", "Dec"]  
financial_Year = [Q01, Q02, Q03, Q04]  
current_Year = financial_Year[:]  
print(current_Year)  
current_Year = list(financial_Year)  
print(current_Year)
```

Output:

```
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']  
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
```

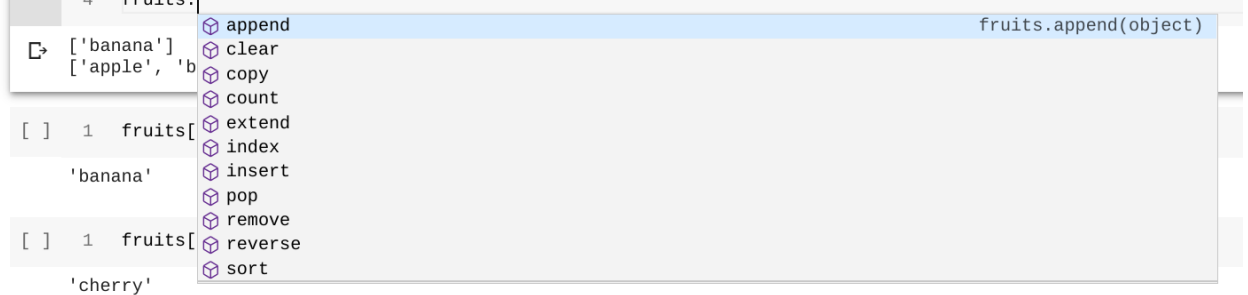
84. Does the expression a_new_list = a_list create a copy of the list a_list?

Ans. Yes, this express will copy all the contents of a_list to a_new_list

85. Where can you find the list of all the methods supported by lists?

Ans. We can access all the methods available and supported by list by using DOT (.) after the name of the list variable. The example is as follows:

```
1 print(fruits[1:2])
2 phal = fruits
3 print(phal)
4 fruits.
```



86. What is a Tuple in Python?

Ans. Advertisements. A tuple is a collection of objects which are ordered and immutable. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Example:

```
week_Days = ('Saturday', 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')
```

```
print(week_Days)
```

Output:

```
('Saturday', 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')
```

87. How is a tuple different from a list?

Ans. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Furthermore the parentheses in tuples are optional but in list these are compulsory. Comma Separated values will be automatically considered as Tuples.

Example: In the following example, both statements can be used to define elements in a tuple.

```
week_Days = ('Saturday', 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')
```

```
week_Days = 'Saturday', 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'
```

88. Can you add or remove elements in a tuple?

Ans. We can't remove any element once defined in a Tuple. Tuples behave like constants in other languages and the value of tuples can't be changed once defined.

89. How do you create a tuple with just one element?

Ans. Creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is, in fact, a tuple.

Examples:

Creating a tuple having one element without using comma (,) will be considered as str

```
my_tuple = ("hello")  
print(type(my_tuple)) # <class 'str'>
```

Creating a tuple having one element

```
my_tuple = ("hello",)  
print(type(my_tuple)) # <class 'tuple'>
```

Parentheses is optional

```
my_tuple = "hello",  
print(type(my_tuple)) # <class 'tuple'>
```

Output:

```
<class 'str'>  
<class 'tuple'>  
<class 'tuple'>
```

90. How do you convert a tuple to a list and vice versa?

Ans. You can convert a Python Tuple to Python List.

`list(sequence)` takes any sequence, in this case a tuple, as an argument and returns a list object.

Example:

In the following example, we initialize a tuple and convert it to list using `list(sequence)`

```
# initialize tuple

aTuple = (True, 28, 'Tiger')

# converting tuple to list

aList = list(aTuple)

# print type of object and list

print(type(aList))

print(aList)
```

Output

```
<class 'list'>
[True, 28, 'Tiger']
```

91. What are the count and index methods of a Tuple used for?

Ans. The `count()` function will count how many times the element appears inside the tuple. On the other hand the function `index()` will return the index of the particular element.

```
# Use of .count() and .index() functions.

my_tuple = ('a', 'p', 'p', 'l', 'e',)

# use o f.count() function

print(my_tuple.count('p'))    # Output: 2

# use o f.index() function
```

```
print(my_tuple.index('1')) # Output: 3
```

Output:

2

3

92. What is a dictionary in Python?

Ans. Python dictionary is an unordered collection of items. Each item of a dictionary has a key/value pair.

Dictionaries are Python's implementation of a data structure that is more generally known as an associative array. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value.

93. How do you create a dictionary?

Ans. We can define a dictionary by using curly braces { } after the name of the dictionary name and equal sign. The data will be defined in a **collection of key-value pairs**.

Example:

```
myself = {  
    'name': 'Shabbir HUssain',  
    'name': 'Abdul Aziz',  
    'sex': 'Male',  
    'age': 42,  
    'education': 'MBA',  
    'married': True  
}
```

More Examples:

empty dictionary

```
my_dict = {}
```

dictionary with integer keys

```
my_dict = {1: 'apple', 2: 'ball'}
```

dictionary with mixed keys

```
my_dict = {'name': 'John', 1: [2, 4, 3]}
```



```
# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

94. What are keys and values?

An item in the dictionary has a **key** and a corresponding **value** that is expressed as a pair (***key: value***).

While the values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

95. How do you access the value associated with a specific key in a dictionary?

While indexing is used with other data types to access values, a dictionary uses **keys**. Keys can be used either inside **square brackets []** or with the **get()** method.

If we use the square brackets [], **KeyError** is raised in case a key is not found in the dictionary. On the other hand, the get() method returns **None** if the key is not found.

```
# get vs [] for retrieving elements
my_dict = {'name': 'Shabbir', 'age': 42}
```

```
# Output: Shabbir
print(my_dict['name'])
```

```
# Output: 42
print(my_dict.get('age'))
```

```
# Trying to access keys which doesn't exist throws error
```

```
# Output None
print(my_dict.get('address'))
```

```
# KeyError
print(my_dict['address'])
```

Output

Shabbir

42

None

Traceback (most recent call last):

File "<string>", line 15, in <module>

print(my_dict['address'])

KeyError: 'address'

96. What happens if you try to access the value for a key that doesn't exist in a dictionary?

If we use the square brackets [], **KeyError** is raised in case a key is not found in the dictionary. On the other hand, the get() method returns **None** if the key is not found.

Trying to access keys which doesn't exist throws error

Output None

```
my_dict = {'name': 'Shabbir', 'age': 42}
```

```
print(my_dict.get('address'))
```

KeyError

```
print(my_dict['address'])
```

Output:

None

Traceback (most recent call last):

File "<string>", line 15, in <module>

print(my_dict['address'])

KeyError: 'address'

97. What is the .get method of a dictionary used for?

.get() method is used to access value from the dictionary. The use of this method is as follows:

Example:

```
# Accessing data using .get() method
my_dict = {'name': 'Shabbir', 'age': 42}
print(my_dict.get('name'))
print(my_dict.get('age'))
```

Output:

```
Shabbir
42
```

98. How do you change the value associated with a key in a dictionary?

Ans. Dictionaries are mutable. We can add new items or change the value of existing items using an assignment operator.

If the key is already present, then the existing value gets updated. In case the key is not present, a new (key: value) pair is added to the dictionary.

Example:

```
# Defining a dictionary
my_dict = {'name': 'Shabbir', 'age': 42}
```

```
# update value
my_dict['age'] = 41
```

```
#Output: {'age': 41, 'name': 'Shabbir'}
print(my_dict)
```

```
# add item
my_dict['address'] = 'Kasur'
```

```
# Output: {'address': 'Kasur', 'age': 41, 'name': 'Shabbir'}
print(my_dict)
```

Output

```
{'name': 'Shabbir', 'age': 42}
{'name': 'Shabbir', 'age': 41, 'address': 'Kasur'}
```

99. How do you add or remove a key-value pair in a dictionary?

We can add new value to the dictionary by defining a new key and value against the same dictionary name. The key will be inside the square braces and the value will be after equal to “=” operator. The system will add the new element at the end of the dictionary.

Example:

```
# Defining a dictionary
my_dict = {'name': 'Shabbir', 'age': 42}

# Printing dictionary before new item was added
print(my_dict)

# add item
my_dict['address'] = 'Kasur'

# Printing dictionary along with new items added.
print(my_dict)
```

Output

```
{'name': 'Shabbir', 'age': 42}
{'name': 'Shabbir', 'age': 42, 'address': 'Kasur'}
```

100. How do you access the keys, values, and key-value pairs within a dictionary?

We can access the key and values and key-value pairs by specifying the dictionary name along with the key inside the square braces. We can print both key and value just using the name of the dictionary inside the print function. The method .get() can also be used to access the key and values from the dictionary.

Example:

```
# Defining a dictionary
my_dict = {'name': 'Shabbir', 'age': 42}

# Printing the whole dictionary along with keys and values
print(my_dict)

# Accessing a specific value from the dictionary
print(my_dict['name'])

# Accessing a specific value from the dictionary using .get() method
print(my_dict.get('name'))
```

Output:

```
{'name': 'Shabbir', 'age': 42}
```

```
'Shabbir'
```

```
'Shabbir'
```

Additional Questions:

1. What is the difference between the * and the ** operators?

Ans. Single Estarisc * operator is used to multiply one number with another and Double Estarisc ** operator is used to calculate exponential power.

Examples:

```
# Use of a single * Operator
```

```
X = 10
```

```
Y = 20
```

```
Z = X * Y
```

```
print("X = ",X)
```

```
print("Y = ",Y)
```

```
print("Multiplication of X & Y = ",Z)
```

```
# Use of a Double ** Operator
```

```
X = 10
```

```
Y = 3
```

```
Z = X ** Y
```

```
print("Power of X & Y = ",Z)
```

Output:

```
X = 10
```

```
Y = 20
```

Multiplication of X & Y = 200

Power of X & Y = 1000

2. What is the order of precedence for arithmetic operators in Python?

Ans. Python works on the fundamental arithmetic operation rules DMAS (Division, Multiplication, Addition and Subtraction)

3. How do you specify the order in which arithmetic operations are performed in an expression involving multiple operators?

Ans. The operators are performed from left to right as per DMAS rule.

4. How do you solve a multi-step arithmetic word problem using Python?

Ans. We can create a program following the arithmetic operation instructions mentioned in word problems to solve it properly.

5. What are variables? Why are they useful?

Ans. A variable is a symbolic name that is a reference or pointer to an object. Once an object is assigned to a variable, you can refer it to the object by that name. The value of the variable can be changed whenever required as per the program requirement. The best use of the variable is when the same value will be required in the program repeatedly/multiple times.

6. How do you create a variable in Python?

Ans. The variables in python can be defined in many ways. Examples are as follows:

X=10

Y="Shabbir Hussain"

Z=10.125

Color1, Color2, Color3 = "Red", "Green", "Blue"

7. What is the assignment operator in Python?

Ans. An Equal to operator "=" is used as an assignment operator to assign any value to the variable.

8. What are the rules for naming a variable in Python?

9. How do you view the value of a variable?

Ans. We can use `print()` command to view the value of the variable.

Example:

```
X=10
Y="Shabbir Hussain"
Z=10.125
print(x)
print(y)
print(z)
```

10. How do you store the result of an arithmetic expression in a variable?

Ans. We can use assignment operators to store the value of an arithmetic expression. The variable name will be on the left side of the operator and the expression will be on the right side.

Example:

```
X=10
Y=13
Z=10.125
ans = x + y - z * y / z
```

11. What happens if you try to access a variable that has not been defined?

Ans. A syntax error will be raised if we try to use a variable that has not been defined yet.

12. How do you display messages in Python?

Ans. `print()` function is used to display messages in Python.

13. What types of inputs can the print function accept?

Ans. Two types of input a `print()` function can accept. First part will be a text message written in the double quotes "" and the second input may be a variable name. If we want to print only the value of the variable name then the variable name will be written in the parentheses () of the print function. If we want to print the value of the variable name along with a text message then the variable name

will be written after the text message written in double quotes separated by a comma (.). The examples are as follows:

Examples:

```
# print a text message only  
print("This is my first program in Python")
```

```
# print the value of a variable only  
name="Shabbir Hussain Bhatti"  
print(name)
```

```
# print text message along with value of the variable  
print("My name is ",name)
```

Output:

```
This is my first program in Python  
Shabbir Hussain Bhatti  
My name is Shabbir Hussain Bhatti
```

14. What are code comments? How are they useful?

Ans. Comments are very important and valuable in any language. These are non executable by the compiler and are used to provide additional information about the code/program.

15. What are the different ways of creating comments in Python code?

Ans. There are two different ways to create a comment in Python Code.

- Single Line Comments

A hash/number signed # symbol is used at the beginning of the comment to create a single line comment in a python code. The example is as follows:

```
# This is an example of a single line comment in python code.
```

- Multi Line Comments

We can write a comment in multiple lines by using three times single quote (' ' ') at the beginning of the comment and also three times single quote (' ' ') at the end of the code as well. The example is as follows:

```
''' This is an example
```


Of multiline comments in
Python code '''

16. What are the different comparison operations supported in Python?

Ans. Followings are the comparison operators supported by Python language to make logical decisions. These operators will always return True or False based on the expression provided.

- Equal to operator ==
- Not Equal to operator !=
- Greater than operator >
- Lesser than operator <
- Greater or Equal to operator >=
- Lessor or Equal to operator <=

17. What is the result of a comparison operation?

Ans. Comparison operators will always return **True** or **False** based on the expression provided.

18. What is the difference between = and == in Python?

Ans. Single Equal to operator (=) is an assignment operator which is used to assign a value to the variable. On the other hand Double Equal Operator (==) is a comparison operator and is used to compare if two values are equal or not. Th examples are as follows:

Examples:

```
# Use of a Single Equal to operator  
name = "Shabbir Hussain Bhatti"  
print(name)
```

```
# Use of a Double Equal to operator  
name == "Shabbir Hussain Bhatti"
```

Output:

```
Shabbir Hussain Bhatti  
True
```

19. What are the logical operators supported in Python?

Ans. The Following are the three logical operators supported in the Python language. These are used to combine two comparison expressions and return **True** or **False** based on the execution of the expression.

- AND Operator
- OR Operator
- NOT Operator

20. What is the difference between the and and or operators?

Ans. The AND operator will return **TRUE** if both expressions used with **AND** operator return **TRUE**. If any of the expressions return **FALSE** the **AND** operator will also return **FALSE**.

The OR operator will return **FALSE** if both expressions used with **OR** operator return **FALSE**. If any of the expressions return **TRUE** the **OR** operator will also return **TRUE**.

AND Operator execution Table is as follows:

AND Operator	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

OR Operator execution Table is as follows:

AND Operator	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

21. Can you use comparison and logical operators in the same expression?

Ans. Yes comparison and logical operators can be used in the same expression but the comparison operators will execute first and then logical operators.

22. What is the purpose of using parentheses in arithmetic or logical expressions?

Ans. Parentheses are used to give priority to the operators in the expression because the arithmetic expressions will follow the DMAS rule. If it is required to execute Addition

and Subtraction before Multiplication and Division then the expression using addition and subtraction operators will be written in the parentheses. The example is as follows:

Examples:

Arithmetic Expression using default DMAS Rule

```
ans = 5 + 10 - 5 * 20 / 2
```

```
print(ans)
```

Output:

-35

Note: This expression will be executed as per the DMAS Rule and the detail is as follows:

```
ans = 5 + 10 - ((5 * 20) / 2)
```

Division and Multiplication will execute first and returns value i.e 5 * 20 will produce 100 and 100 / 2 will produce 50

Addition and Subtraction will be executed after multiplication and division. Now the new expression will be like this 5 + 10 - 50. Now addition will execute first and produce 5 + 10 = 15 which will be subtracted from 50 i.e 15 - 50 and the final result will be -35.

Arithmetic Expression using parentheses

```
ans = (5 + 10 - 5) * 20 / 2
```

```
print(ans)
```

Output:

100

Note: This expression will be executed with a priority expression written inside the parentheses. The expression written in parentheses (5+10-5) will produce +10 result and the new expression will be = 10 * 20 / 2. Now 10 * 20 will produce 200 which will be divided by 2 and produce the final result as (200 / 2) 100.

How to Delete a Tuple?

Ans. As we know that we cannot change/modify the elements in a tuple. It means that we cannot delete or remove items from a tuple. But we can delete a tuple entirely by using the keyword **del**.

```
# Deleting tuple
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# can't delete items
# TypeError: 'tuple' object doesn't support item deletion
# del my_tuple[3]

# Can delete an entire tuple
del my_tuple

# NameError: name 'my_tuple' is not defined
print(my_tuple)
```

Output:

Traceback (most recent call last):

```
File "<string>", line 12, in <module>
NameError: name 'my_tuple' is not defined
```

How to combine two different tuples?

Ans. We can use + operator to combine two tuples. This is called concatenation. We can also repeat the elements in a tuple for a given number of times using the * operator.

Both + and * operations result in a new tuple.

Examples:

```
# Concatenation using "+" operator
print((1, 2, 3) + (4, 5, 6))
```

```
# Repeat using "*" operator
print(("Repeat",) * 3)
```

Output:

```
(1, 2, 3, 4, 5, 6)
('Repeat', 'Repeat', 'Repeat')
```

How to check if an element exists inside the tuple?

Ans. We can test if an item exists in a tuple or not, using the keyword **in**. We can also use the Not **in** operator to check that the element does not exist by getting the answer True or False. Keep in mind if the answer is TRUE then it means the item does not exist in the tuple. The example is as follows:

```
# Membership test in tuple using in & Not in Operator
```

```
my_tuple = ('a', 'p', 'p', 'l', 'e',)
```

```
print('a' in my_tuple)
```

```
print('b' in my_tuple)
```

```
# Not in operation
```

```
print('g' not in my_tuple)
```

Output

```
True
```

```
False
```

```
True
```

How to Remove Elements from a Dictionary?

We can remove a particular item in a dictionary by using the **pop()** method.

This method removes an item with the provided **key** and returns the **value**.

The **popitem()** method can be used to remove and return an arbitrary (**key**, **value**) item pair from the dictionary. All the items can be removed at once, using the **clear()** method.

We can also use the **del** keyword to remove individual items or the entire dictionary itself.

Example:

```
# Removing elements from a dictionary
```

```
# create a dictionary
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# remove a particular item, returns its value
print(squares.pop(4))
# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)

# remove an arbitrary item, return (key,value)
print(squares.popitem())
print(squares)

# remove all items
squares.clear()
print(squares)

# delete the dictionary itself
del squares

# Throws Error
print(squares)
```