DOMS | Page No.
Date / /

Python Question Bank.

Q. 1.

Ans a) Python default arguments

=> i) Python allows function arguments to have default values; if the function is called without the argument, the argument get its default value.

ii) The default value is assigned by using assignment (=) operator.

b) Keyword

iii)
```
def DefaultArg (name, foo = 'come here!'):
    print(name, foo)
DefaultArg ('Yash')
```

Output : Yash come here!

b) Keyword Argument

=> i) Keyword arguments are related to the function calls. When you use keyword argument in a function call, the caller identifies the arguments by the parameter name.

ii)
```
def printme(str):
    print str
    return
printme(str = 'Yash')
```
Output : Yash.

DOMS | Page No.
Date / /

c) Python Arbitrary Argument
i) Sometimes, we do not know in advance the number of arguments that will be passed into a function. Python allows us to handle this kind of situation through function calls with an arbitrary number of arguments.
ii) We use an asterisk (*) before the parameter name to denote this kind of argument.

```
def greet (*names):
    for name in names:
        print ("Hello", name).

greet ("Monica", "Luke", "Steve", "John").
```

output,
Hello Monica
Hello Luke
Hello Steve
Hello John

d) Python Positional Argument
i) A positional argument in Python is an argument whose position matters in a function call.

```
def info(name, age)
    print (f"Hi, {name}. {Age}")

info ("Alice", 23) # allowed. allow
info (30, "Alice") # not allow.
```

DOMS | Page No.
Date / /

Q.2)

Ans) A) i) A module allows you to organize your python code. Grouping related into a module makes the code easier to understand and use. A module is a python object with arbitrarily named attributes that you can bind and reference.

ii) A module can define functions classes and variables. A module also include runnable code.

Let us create a python module ~~aname.py~~ support.py which consists of block of code which is shown below.

```
def print_func(par):
    print("Hello : ", par)
    return
```

Now we will create another file ~~which will name~~ aname.py in this we will import support.py.

```
import support
support.print_func("Zara")
```

Output: Hello : Zara.

B) i) A package is a hierarchical file directory structure that defines a single Python application environment that consists of module and ~~sub~~ subpackages and so on.

ii) Creating a Package.

(1) Create a folder name mypkg.

(2) Inside this folder create an empty python file i.e __init__.py

(3) Then create two modules mod1 and mod2 in this folder.

mod1.py

```
def gfg():
    print("Hello").
```

mod2.py

```
def sum(a, b):
    return a+b.
```

mypckg

- __init__.py
- mod1.py
- mod2.py

__init__.py

```
from .mod1 import gfg.
from .mod2 import sum.
```

Now we will import the module from the above created package and will use the function inside those modules.

```
from mypckg import mod1.
from mypckg import mod2
mod1.gfg()
res = mod2.sum(1, 2)
print(res)
```

Output:

Yash.

3 //

Q3.)

One) (1) Arithmetic Operators perform various arithmetic calculation like addition, substraction, multiplication division, modulus, exponent etc.

```
x = 5, y = 6
print(x+y)
```

(2) Comparison Operator in Python compares the value on either side of the operand and determines the relation blw them.

(e.g. <=, !=, <>, >=, etc.)

```
x = 4
y = 5
print ('x>y is', x>y))
```

(3) Assignment Operator in python are use for assigning the value of the right operand to the left operand

```
num1 = 4
num2 = 5
print(num1, num2)
```

} Example

(4) Logical Operator

Logical Operator in Python are use for conditional statements are true or false.

And → Both are true then true

OR → Both are false then false

Not → returns true if operand is false.

Eg a = True

b = False

print (a and b)

Output : false.

(5) Membership Operator
These operator test for membership in a sequence such as list, strings, or tuples.

(6) Identity Operator.
Identity operator are used to compare the objects; not if they are equal, but if they are actually same object, with the same memory location.
is and is not are identity operator.

```
x = ['apple']
y = ['apple']
z = x
print (x is z)
```
output :- True

Q.4)
Ans) Different types of data structure in python are:-

(1) Lists :- Lists are use to store data of different type in a sequential manner. There are addres assigned to every element of list, which is called as index.

Example :-
```
my_list = [1, 2, 3, 'example']
print (my_list)
```

Output :- [1, 2, 3, 'example']

(2) Dictionary :- Dictionary is used to store key - value pairs. To understand better,

Eg.
my_dict = {1: 'python', 2: 'Java'}
print (my_dict).
{1: 'python', 2: 'Java'}

(3) Tuple:
Tuple are the same as list are with the exception that data once entered into the tuple cannot be change no matter what.
Ex my_tuple = (1, 2, 3).
print (my_tuple).
output = (1, 2, 3).

(4) Sets
Sets are collection of unordered elements that are unique. Meaning that even if the data is repeated more than one time it would be entered into the set only once.

my_set = {1, 2, 3, 4, 5, 5, 5}
print (my_set)
output {1, 2, 3, 4, 5}.

Q.5)
Ans) (1) List
i) my_list = [1, 2, 3]
my_list.append (4)
my_list
output :- [1, 2, 3, 4]

DOMS | Page No.
Date / /

ii] extend.
(i) my_list.extend([5,6])
my_list
output :- [1, 2, 3, 4, 5, 6].

iii] insert
my_list.insert(3,7)
my_list
Output :- [1, 2, 3, 7, 4, 5, 6].

iv] remove.
my_list.remove(7)
my_list
output :- [1, 2, 3, 4, 5, 6]

v] pop
my_list.pop(i)
my_list
output :- [1, 2, 3, 4, 5].

vi] slice.
print(my_list[2:])
output :- [3, 4, 5]

vii] reverse.
print(my_list.reverse())
[5, 4, 3, 2, 1] :- output

viii] len.
print(len(my_list))
output :- 5.

count

ix] print(my_list.count(3))
Ouput:- 1.

x] Index.
print(my_list.index(3))
output:- 2

(2) Tuple
i] mt = (1, 2, 3, 4)
mt = mt + (5,)
print(mt)
Output:- (1, 2, 3, 4, 5)

ii] Index
print(mt[1])
Output:- 2

iii] delete
del (mt)
Note:- Tuple is similar to list so almost of the operation of list and tuples are same.

(3) Dictionary
i] my_dict = {"Name": [], "Address": [], "Age": []};
my=dict.append.
my_dict["Name"].append("Yash")
my=dict.app
my_dict["Address"].append("Mumbai").
my_dict["Age"].append(19)
print(my_dict)
output:- {'Name': ['Yash'], 'Address': ['Mumbai'], 'Age': [30]}

DOMS | Page No.
Date / /

Insert.

ii] my_dict["height"] = 5.8
output:
{ — 11 ——, height : 5.8 }.

iii] reverse
res = dict(reversed(list(my_dict.items())))
print(str(res))

iv] len
len(my_dict)

v] dd slice
dict(itertools.islice(d.items(), 2))

(4) Sets

i] ms = {1, 2, 3, 4}
ms.add(5)

ii] ~~res~~ Pop.
ms.pop()

iii] ~~ms1~~ = {5, 6, 7} Union
ms1 = {5, 6, 7}
ms2 = ms.union(ms1)
print(ms2).

iv] Remove
ms.remove(5)

v] Copy
ms2.copy().

DOMS | Page No.
Date / /

Q:5

Ans) (1) Python docstrings are the string literal that appears right after the definition of a function, method, class, or module. Lets take an example.

(2) The docstring is use to document a specific segment of code. The docstring for any particular python object can be accessed by using a doc attribute for a object.

(3) E.g : How we can access. docstring.

```
def square(n)
'''Takes in a number n, returns the square of n'''
    return n**2
print(square.__doc__)
```

Output :- Takes in a number n, returns the square of n.

Q.6)

Ans) A) Membership Operator

i) Membership operator are operators use to validate the membership of a value. It test for membership in a sequence, such as string, list or tuples.

ii) 'in' operator is use to check if a value exist in a sequence or not. Evaluate to true if it finds a variable in the specified sequence and false otherwise.

E.g

```
list1 = [1, 2, 3]
list2 = [4, 5]
for item in list1:
    if item in list2:
        print("overlapping").
else:
    print("not overlapping").
```

Output :- not overlapping.

iii) 'not in' operator evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

e.g x = 24
list = [10, 20, 30, 40]
if (x not in list):
    print ("Happy")
else:
    print ("lol").
Output: Happy

B) Identity Operator

i) In python identity operator are use to determine whether a value is of a certain class or type.

ii) 'is' operator evaluates to true if the variable on either side of the operator point to the same object and false otherwise.

e.g x = 5
if (type(x) is int):
    print ('True')
else:
    print ('False')
Output: True.

iii) 'is not' operator - Evaluates to false if the variable on either side of the operator point to a different object and true otherwise.

x = 5
if (type(x) is not int):
    print ('True')
else:
    print('False')
Output: 'False.

DOMS | Page No.
Date / /

Q.7

Ans)

Bitwise Operator	Logical Operator
i) &	i) &&
ii) Operates on bit values	ii) operates on boolean expression.
iii) both sides will get evaluated as not true/false case is here.	iii) If first expression return false, it will conclude the answer and false and will ignore second expression.
iv) use in bit manupulation	iv) use in loops and conditional.
Example:	Example:
4 & 6 = 6	if ((1+2==6) && (3+2==5))
(a) 0 0 0 0 0 1 0 0	sout ("hello !!");
0 0 0 0 0 1 1 0	Hello will not get printed as 1st
0 0 0 0 0 1 0 0	exp will result in false, which will make entire expression false.

DOMS | Page No.
Date / /

Q.8)

Ans: i) Already written about module in Question 2.

ii) A function is a block of organized, reusable code that is used to perform a single related action.

iii) a) Function blocks begin with the keyword def followed by the function name and parentheses(()).

b) any input parameters or arguments should be place within this parantheses.

c) Can include docstring it is optional.

d) The code block within every function starts within a colon (:) and is indented.

e) The statement return exits a function, optionally passing back an expression to the caller.

Syntax.

```
def functionname(parameters):
    "function_docstring"
    function_suite
    return [expression]
```

Example

```
def printme(str):
    "this prints a passed string into this func"
    print str
    return
```

DOMS | Page No.
Date / /

Q.9)

Ans) (1) The pass statement is use as a placeholder for future code.

(2) When the pass statement is executed, nothing happens, but you avoid getting an error. When empty code is not allowed.

(3) Empty code is not allowed in loops, function definitions, class definition, or in if statements.

(4) Syntax Example:

```
def myfunc():
    pass
```

Q10)

Ans) i) Python string is a sequence of Unicode characters that is enclosed in the quotation marks.

ii) Every strings method does not change the original string instead returns a new string with the change attribute.

iii) Example

```
my_string = 'Hello'
print (my_string)
```

output: Hello

iv) String the op

v) There are many operations that can be performed with strings which make it one of the most use data types in python.

a) Concatenation of Two or More Strings.

i) Joining of two or more strings into a single one is called concatenation.

ii) The + operator does this in python.

iii) the * operator can be used to repeat the strings for a given number of times.

iv) str1 = 'Hello'
str2 = 'World'
print (str1 + str2)
print (str1 * 3)
Output:- HelloWorld
HelloHelloHello

v) If we want to concatenate strings in different line we can use parentheses.
s = ('Hello'
'World')
print(s)
Output:- 'Hello World'

b) Iterating through String
i) We can iterate through a string using a for loop
ii) count = 0
for letter in 'Hello World':
if(letter == 'l'):
count += 1
print(count, 'letters found')
Output:- 3 letters found.

c) String Membership Test
i) We can test if a substring exist within a string or not, using the keyword in.

DOMS | Page No.
Date / /

ii) 'a' in 'program'
Output :- True
'at' not in 'battle'
Output :- False.

d) Built-in functions to work with python:
a i) various built-in functions that work both sequence work with string as well. The enumerate() functions returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.
ii) len() returns the length of the strings.
iii) str = 'cold'
list_enumerate = list(enumerate(str)) # enumerate.
print(list_enumerate).
print(len(str)) # Character Count

Output :- [(0, 'c'), (1, 'o'), (2, 'l'), (3, 'd')]
4

Q.11)
Ans) i) The str.split() method is use to split strings up.
ii) book = 'Problem Solving and Python Programming'.
print(book.split())
Output :- ['Problem', 'Solving', 'and', 'Python', 'Programming']

Q.12)
Ans i) The elements of a dictionary appear in comma-seperated list. Each entry contains an index and a value seperated by a colon. In a dictionary, the indices are called keys, so the elements are called key value pairs.

ii) Some of the operations on dictionary are:-

a) del.

```
y = {'one': 1, 'two': 2}
del y['two']
print(y)
```

Output :- {'one': 1}

b) Update

```
x = {'one': 0, 'two': 2}
y = {'one': 1, 'three': 3}
x.update(y)
print(x)
```

Output

{'one': 1, 'two': 2, 'three': 3}

c) keys.

```
x = {'one': 1, 'two': 2}
print(x.keys())
```

Output :- dict_keys(['one', 'two'])

d) values.

```
x = {'one': 1, 'two': 2}
print(x.values())
```

Output :- dict_value([1, 2])

e) items

```
x = {'one': 1, 'two': 2}
print(x.items())
```

Output :- dict_items([('one', 1), ('two', 2)])

DOMS | Page No.
Date / /

Q13)

Ans

	Break	Continue
①	Break terminates the execution of remaining iteration of the loop	① Continue terminates only the current iteration of the loop
②	Break resumes the control of the program to the end of loop enclosing that 'break'	② Continue resumes the control of the program to the next iteration of that loop enclosing 'continue'.
③	It causes early termination of loop.	③ It causes early execution of the next iteration.
④	Keyword used is "break"	④ Keyword used is "continue"
⑤	Example:-	⑤ Example:-

```
for i in range(3):
    if i==2:
        break
    else:
        print(i)
```

Output:-
0
1

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

Output:-
0
2

Q14)

Ans. a) The range() function is used to generate a sequence of numbers over time or it is used when a user.

Q.14)

ans a) Range() is used when a user needs to perform an action a specific number of times.

Syntax

range(start, stop, step)

Example:-

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

Output:

2
4
6
8

b) Using range in a list: first of all we cannot directly use range function in a list. so we can use following which has given below.

a) We can use argument-unpacking operator i.e *.

```
my_list = [*range(2, 10, 2)]
print(my_list)
```

output: [2, 4, 6, 8]

b) We can use the extend() function to unpack the result of range function.

```
my_list = []
start, end = 2, 6
if start < end:
    my_list.extend(range(start, end))
    my_list.append(end)
print(my_list)
```

Output:

[2, 3, 4, 5]

c) The range function in python is a function that let us generate a sequence of integer values lying between a certain range. The function also let us generate these

values with specific step value as well. It is represented by the equation or syntax:-
range (start, stop, step).
Unlike range function, arange function in python is not a built in function. But instead, it is a function we can find in the Numpy module. So in order for you to use the arange function, you will need to install Numpy package first!
As arange function looks like this:
numpy.arange (start, stop, step, dtype = None)
The built in range function can generate only integer values that can be accessed as list elements. But on the other hand, arange function can generate values that are stored in Numpy arrays. We can observe this in the following code:

```
import numpy as np
a = np.arange(4)
print(a)
```

~~array [0, 1, 2, 3])~~
~~Output: [0, 1, 2,~~
output: [0 1 2 3]

The range function is considerably slower. So if speed matters to you, use arange function.

Q.15)
Ans) i) grid()
ii) The grid() geometry manager organizes the ~~widgets~~ widgets in the tabular form. We can specify the rows and column as the option in the method call. We can also specify the column span (width) or row span (height) of a widget.
ii) This is a more organized way to place the widgets

to the python application. The syntax to use the grid() is given below.
syntax :- widget.grid(options).

iii) A list of possible options that can be passed inside the grid() method is given below:

a) Column :-
The column number in which the widget is to be placed. The leftmost column is represented by 0.

b) columnspan :-
The width of the widget. It represents the number of columns up to which, the column is expanded.

c) ipadx, ipady :-
It represent the number of pixels to pad the widget inside the widget's border.

d) padx, pady :-
It represents the number of pixels to pad the widget outside the widget's bolder.

e) row :-
The row number in which the widget is to be placed. The topmost row is represented by 0.

f) rowspan :-
The height of the widget, i.e the number of the row up to which the widget is expanded.

g) sticky :-
If the cell is larger than a widget, then sticky is used to specify the position of the widget inside the cell.

(2) place()

i) The place() geometry manager organizes the widgets to the specific x and y coordinates.

ii) Syntax :- widget.place(option).

ii) A list of possible option is given below.

a) Anchor :- It represents the exact position of the widget within the container. The default value (direction) is NW (the upper left corner).

b) bordermode :- The default value of the border type is ~~outside~~ INSIDE that refers to ignore the parent's inside the border. The option is OUTSIDE.

c) height, width :- It refers to the height and width in pixels.

d) relheight, relwidth :- It is represented as the float b/w 0.0 and 1.0 indicating the fraction of the parent's height and width.

e) relx, rely :- It is represented as the float b/w 0.0 and 1.0 that is the offset in the horizontal and vertical direction.

f) x, y :- It refers to the horizontal and Vertical offsets in the pixels.

(B) Pack()

i) (A) the pack() widget is used to organize widget in the block. The position widgets added to the python application is done using the pack() method can be controlled by using the various options specified in the method call.

ii) (B) Syntax :- widget.pack(options)

iii) A list of possible option that can be passed in pack() is given below.

a) expand :- If the expand is set to true, the widget expands to fill any space.

b) Fill :- By default, the fill is set to NONE. However, we can set it to X and Y to determine whether the widget contains any extra space.

c) Size :- It represents the side of the parent to which the the widget is to be placed on the window.

Q.16)

Ans) A) Features of Python Programming are :-

a) Easy to code :-

Python is high-level programming language. Python is very easy to learn the language as compared to other language like C, C#, Javascript, Java and c++. etc. Anybody can learn python basics in few hours or days. It is also developer friendly prog. language.

b) Embeddable :-

The code of the other programming language can use in the python source code. We can use Python source code in another programming language as well. It can embed other language into our code.

c) Interpreted Language :-

Python is an interpreted language, it means the Python program is executed one line at time. The advantage of being interpreted language, it makes debugging easy and portable.

d) Object-Oriented Language :-

Python supports object-oriented language and concepts of classes and objects come into existence. ~~It supports~~ The object oriented procedure helps to programmer to write reusable code and develop application in less code;

e) Free and Open Source :- &

Python is freely available for everyone on its official website www.python.org. The open source means,

"Anyone can download its source code without paying any penny."

f) GUI programming Support :-
GUI is used for the Developing Desktop application. PyQT5, Tinter, Kivy are the libraries which are used for developing the Web Application.

B) Python is very popular because it has simplified syntax and not complicated, which gives more emphasis on natural language. Due to its ease of learning and usage, python codes can be easily written and executed much faster than other programming languages.

Q.17)

Ans) a) append() - add an element to end of list
b) insert() - insert an item at the defined index
c) remove() - removes an item from the list
d) clear() - removes all item from the list
e) reverse() - reverse the order of items in the list

Q.18)

Ans) ① Tuple is used for heterogeneous data types and list is used for homogenous data types.
② Since tuple are immutable, iterating through tuple is faster than list.
③ Tuple that contain immutable elements can be used as key for dictionary.
④ Implementing data that doesn't change as a tuple remains write-protected.

Q.19 In the.

Ans) i) Given command, tuple[1:3] is accessing the items in the tuple using indexing. It will print elements starting

from 2nd till 3rd.

ii) Output will be (786, 2.23)

Q.20

Ans) (1) If you wont to modify a dictionary and keep a copy of the original, use the copy method.

(2) For example, opposites is a dictionary that contain pairs of opposites:

```
opposites = {'up': 'down', 'right': 'wrong', 'true': 'false'}
alias = opposites
copy = opposites.copy()
```

alias and opposites refer to the same object; copy refers to a fresh copy of the same dictionary. If we modify alias, opposites is also changed:

```
alias['right'] = 'left'
opposites['right']
```

» 'left'

(3) If we modify copy, opposites is unchanged:

```
copy['right'] = 'privilege'
```

» 'left'

Q.21

Ans) (1) List and Tuple objects are sequences. A dictionary is a hash table of key-value pairs. List and Tuple is a data an ordered collection of items. Dictionary is unordered collection.

(2) List and dictionary objects are mutable i.e it is possible to add new item or delete and item from it. Tuple is an immutable object. Addition or deletion operations are not possible on Tuple object.

(3) Each of them is a collection of comma seperated items. List items are enclosed in square brackets [] tuple item in round brackets or parentheses (), and dictionary item in curly brackets {}.

(4)
```
L1 = [12, "Yash", "CSE", 9.00] # list
T1 = (12, "Yash", "CSE", 9.00) # tuple
D1 = {"RollNo": 12, "Name": "Yash", "Course": "CSE", "marks":
    9.00} # dictionary
```

(5) List and tuple items are indexed. Slice operator allows item of certain index to be accessed.
```
print(L1[2])
CSE
print(T1[2])
CSE
```

(6) Items in dictionary are not indexed. Value associated with a certain key is obtained by putting in square ~~bracket~~ bracket. The get() method of dictionary ~~also~~ also returns associated ~~value~~ value.
```
print(D1['Course'])
CSE
print(D1.get('Course'))
CSE.
```

Q.22

Ans A) (1) The write() method writes a specified text to the file. Where the specified text will be inserted depends on the file mode and stream position.
"a": The text will be inserted at the current file stream position, ~~def~~ default at the end of the file.
"w": The ~~file~~ file will be emptied before the text will be inserted at the current file stream position, default 0

DOMS | Page No.
Date / /

(2) Syntax :- file.write(byte).
byte :- The text or byte object that will be inserted.
Example.
f = open("demo.txt", "a")
f.write("Hi")
f.close()

# open and read the file after the appending :
f = open("demo.txt", "r")
print(f.read())

Output :- Hi

(B) The only difference between the write() and writelines() is that write() is used to write a string to an already opened file while writelines() method is used to write a list of string in an opened file.