GUIDELINES FOR VALUE ADDED COURSE (VAC) 2021-22

1. Evaluation

The value added courses shall carry 100 marks and shall be evaluated through internal Assessments only.

Continuous Assessment (CA)

The CA shall be a combination of a variety of tools such as class test, assignment, seminars, and viva-voce that would be suitable to the course.

The break-up of marks shall be as follows:

Theory Course

Item	Marks	Grading Marks
Quiz Tests/Class Assignments/Home Assignments/**Google form online test**	40	4
Seminar/ Class Presentations /Class Performance	30	3
Viva-voce	30	3
Total	100	10

Practical Course

Item	Marks	Grading Marks
Demonstration of Skills and Viva Voce	40	4
Assignments& Exercises	30	3
Lab Performance	30	3
Total	100	10

Continuous Assessment Tests

i. Continuous assessments shall be conducted preferably one in the middle and other at the end of the course.

ii. The duration of the test, the pattern of question paper and the units included shall be decided by the Course Coordinator and prior intimation shall be given to the students.

iii. The assessment shall be done by the course teacher/Course Coordinator.

iv. No improvement option shall be available for CA. However, if a student could not attend the test for any valid reason, the prerogative of arranging a special test lies with the Course Coordinator in consultation with the Head of the Department.

2. Grading

Evaluation of the performance of the student will be rated as shown in the Table

Marks	Grade	Grade Point
90 – 100	O	10
75 – 89	A+	9
65 – 74	A	8
55 – 64	B+	7
50 – 54	B	6
45 – 49	C	5
40 – 44	P	4
Less than 40 or absent	F	0

The grades and credits obtained in VACs shall not be considered for calculating the GPA and CGPA of the regular course that the student is undergoing. The percentage of marks obtained by a candidate in a course will be indicated in the awarding certificate.

3. Awarding Certificate

On successful completion of the VAC, the student shall be issued a certificate duly signed by the Head of the Department and the Course Coordinator.

Course Coordinator **HoD**

Assessment

Ethical Hacking

* Indicates required question

1. Email *

2. Email *

3. Name *

4. Roll No. *

5. 1) What built-in list method would you use to remove items from a list? 1 point

Mark only one oval.

- .delete() method
- pop(my_list)
- del(my_list)
- .pop() method

6. 2) What is the purpose of an if/else statement? 1 point

Mark only one oval.

- ◯ It tells the computer which chunk of code to run if the instructions you coded are incorrect.
- ◯ It runs one chunk of code if all the imports were successful, and another chunk of code if the imports were not successful.
- ◯ It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.
- ◯ It tells the computer which chunk of code to run if the is enough memory to handle it, and which chunk of code to run if there is not enough memory to handle it.

7. 3) What does the built-in map() function do? 1 point

Mark only one oval.

- ◯ It creates a path from multiple values in an iterable to a single value.
- ◯ It applies a function to each item in an iterable and returns the value of that function.
- ◯ It converts a complex value type into simpler value types.
- ◯ It creates a mapping between two different elements of different iterables.

8. 4) If you don't explicitly return a value from a function, what happens? 1 point

Mark only one oval.

- ◯ The function will return a RuntimeError if you don't return a value.
- ◯ If the return keyword is absent, the function will return None.
- ◯ If the return keyword is absent, the function will return True.
- ◯ The function will enter an infinite loop because it won't know when to stop executing its code.

9. 5) When does a for loop stop iterating? 1 point

Mark only one oval.

- when it encounters an infinite loop
- when it encounters an if/else statement that contains a break keyword
- when it has assessed each item in the iterable it is working on or a break keyword is encountered
- when the runtime for the loop exceeds O(n^2)

10. 6) What is key difference between a set and a list? 1 point

Mark only one oval.

- A set is an ordered collection unique items. A list is an unordered collection of non-unique items.
- Elements can be retrieved from a list but they cannot be retrieved from a set.
- A set is an ordered collection of non-unique items. A list is an unordered collection of unique items.
- A set is an unordered collection unique items. A list is an ordered collection of non-unique items.

11. 7) Review the code below. What is the correct syntax for changing the price to 1.5? 1 point

```
fruit_info = { 'fruit': 'apple', 'count': 2, 'price': 3.5 }
```

Mark only one oval.

- fruit_info ['price'] = 1.5
- my_list [3.5] = 1.5
- 1.5 = fruit_info ['price]
- my_list['price'] == 1.5

12. 8) You are given a piece of code. Assume m and n are already defined as some positive integer value. When it completes, how many tuples will my list contain? 1 point

```
mylist = []
for i in range(m):
    for j in range(n):
        mylist.append((i,j))
```

Mark only one oval.

- m
- m + n
- n
- m * n

13. 9) Assume m, n and p are positive integers. In the following comprehension, how many times will the function randint be called? 1 point

```
[ [ [ randint(1,100) for i in range(m) ] for j in range(n) ] for k in range(p) ]
```

Mark only one oval.

- m _ n _ p
- the greater value of (m,n,p)
- 1 million
- m + n + p

14. 10) What is the correct syntax for replacing the string apple in the list with the string orange? 1 point

my_list = [2, 'apple', 3.5]

Mark only one oval.

- orange = my_list[1]
- my_list[1] = 'orange'
- my_list['orange'] = 1
- my_list[1] == orange

15. 11) Which comparison of lists and tuples in Python is correct? 1 point

Mark only one oval.

- Use lists instead of tuples when you have a collection of related but dissimilar objects.
- Use tuples instead of lists when you have a common collection of similar objects.
- Use tuples instead of lists for functions that need to return multiple values.
- Use lists instead of tuples when the position of elements is important.

16. 12) What will this code output to the screen? 1 point

```
for i in range(5):
  print(i)
else:
  print("Done!")
```

Mark only one oval.

- 1 2 3 4 5 Done!
- 0 1 2 3 4 5 Done!
- 0 1 2 3 4 Done!
- You will get a syntax error.

17. 13) Which collection is ordered, changeable, and allows duplicate members? 1 point

Mark only one oval.

- SET
- TUPLE
- DICTIONARY
- LIST

18. 14) What happens if the file is not found in the following Python code? 1 point

```
a=False
while not a:
    try:
        f_n = input("Enter file name")
        i_f = open(f_n, 'r')
    except:
        print("Input file not found")
```

Mark only one oval.

- No error
- Assertion error
- Input output error
- Name error

19. 15) What will be the output of the following Python code? 1 point

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

Mark only one oval.

- NameError
- ValueError
- IndexError
- TypeError

20. 16) Identify the type of error in the following Python codes? 1 point
Print(“Good Morning”)
print(“Good night)

Mark only one oval.

- Syntax, Syntax
- Semantic, Syntax
- Semantic, Semantic
- Syntax, Semantic

21. 17) An exception is _____ 1 point

Mark only one oval.

- an object
- a special function
- a standard module
- a module

22. 18) How many keyword arguments can be passed to a function in a single function call? 1 point

Mark only one oval.

- zero
- one
- zero or more
- one or more

23. 19) How many except statements can a try-except block have? 1 point

Mark only one oval.

- 0
- 1
- more than one
- more than zero

24. 20) Which of the following will print the pi value defined in math module? 1 point

Mark only one oval.

- print(pi)
- print(math.pi)
- from math import pi print(pi)
- from math import pi print(math.pi)

25. 21) Which operator is used in Python to import modules from packages? 1 point

Mark only one oval.

- .
- *
- ->
- &

26. 22) How is a function declared in Python? 1 point

Mark only one oval.

- def function function_name():
- declare function function_name():
- def function_name():
- declare function_name():

27. 23) Which one of the following is the correct way of calling a function? 1 point

Mark only one oval.

- function_name()
- call function_name()
- ret function_name()
- function function_name()

28. 24) Choose the correct option with reference to below Python code? 1 point

```
def fn(a):
    print(a)

x=90

fn(x)
```

Mark only one oval.

- x is the formal argument.
- a is the actual argument.
- fn(x) is the function signature.
- x is the actual argument.

29. 25) Which one of the following is incorrect? 1 point

Mark only one oval.

- The variables used inside function are called local variables.
- The local variables of a particular function can be used inside other functions, but these cannot be used in global space
- The variables used outside function are called global variables
- In order to change the value of global variable inside function, keyword global is used.

Email Address	Score	Email
priyanjalmalhotra4@gmail.com	23 / 25	priyanjalmalhotra4@gmail.com
gopalrajput11a@gmail.com	24 / 25	gopalrajput11a@gmail.com
singhal2002y@gmail.com	9 / 25	singhal2002y@gmail.com
siddharthpopli15@gmail.com	24 / 25	Siddharthpopli15@gmail.con
rohittanwar1287@gmail.com	20 / 25	rohittanwar1287@gmail.com
vrindasuneja2004@gmail.com	18 / 25	vrindasuneja2004@gmail.com
prajjwaldwivedi01@gmail.com	9 / 25	prajjwaldwivedi01@gmail.com
viveksharma8481@gmail.com	18 / 25	8750779948
keshavkumar691999@gmail.com	23 / 25	9205490649

thakurharsh236@gmail.com	21 / 25	8938810595
deepakaiml12@gmail.com	17 / 25	9671248852
gibrailzaidi@gmil.com	17 / 25	8126560284
gazabthakur07@gmail.com	17 / 25	8958280167
adityamishra.am71@gmail.com	23 / 25	9835319964
bhinderfazal@gmail.com	23 / 25	90454097615
anujmishra7069@gmail.com	23 / 25	8521407255
adarshrajput1914@gmail.com	17 / 25	adarshrajput1914@gmail.com
bhardwajsagar5498@gmail.com	11 / 25	bhardwajsagar5498@gmail.com
choudharyanas19@gmail.com	17 / 25	8920295476

hs9650612804@gmail.com	21 / 25	9650612804
faizannasim59@gmail.com	20 / 25	7532053343
md.amberkhan9631@gmail.com	15 / 25	8700753024
Srivastavayushbth@gmail.com	17 / 25	6205646148
gaganshuyadav16@gmail.com	21 / 25	9868043151
faridianzar786@gmail.com	20 / 25	7895829079
arunpandey916162@gmail.com	17 / 25	8756295300
anshumansoni4546@gmail.com	17 / 25	7237031490
himanshusharma1454@gmail.com	17 / 25	6393381416
vermavishesh345@gmail.com	16 / 25	vermavishesh345@gmail.com

kamranahmadmd463@gmail.com	20 / 25	kamranahmadmd463@gmail.com
viraz.rr@gmail.com	16 / 25	viraz.rr@gmail.com
manishking241@gmail.com	18 / 25	9177893190
shivamsinghcse19@gmail.com	20 / 25	shivamsinghcse19@gmail.com
himanshurautela87@gmail.com	22 / 25	himanshurautela87@gmail.com
rawatsumit1290@gmail.com	23 / 25	rawatsumit1290@gmail.com
hs185008@gmail.com	21 / 25	hs185008@gmail.com
me.souravk03@gmail.com	7 / 25	me.souravk03@gmail.com
arcuscode@gmail.com	15 / 25	arcuscode@gmail.com
naveensangwanhas@gmail.com	15 / 25	naveensangwanhas@gmail.com

vaibhavjain98111@gmail.com	15 / 25	vaibhavjain98111@gmail.com
fagunchauhan4@gmail.com	16 / 25	fagunchauhan4@gmail.com
devmohansharma177013@gmail.com	19 / 25	devmohansharma177013@gmail.com
sidra.tabassum2004@gmail.com	12 / 25	sidra.tab1804@gmail.com
jhanvikhanna2@gmail.com	13 / 25	jhanvikhanna2@gmail.com
aviralrastogi2004@gmail.com	13 / 25	aviralrastogi2004@gmail.com
aanchalmangla73@gmail.com	19 / 25	aanchalmangla73@gmail.com
rajpriya012005@gmail.com	20 / 25	rajpriya012005@gmail.com
tcccvv@gmail.con	6 / 25	Yfgb@ynmq
summitparmar@gmail.com	23 / 25	summitparmar@gmail.com

PARTHG5082@GMAIL.COM	22 / 25	parthg5082@gmail.com
mittalmanya10@gmail.com	17 / 25	mittalmanya10@gmail.com
mrukaiya13@gmail.com	24 / 25	mrukaiya13@gmail.com
sagarparasher04@gmail.com	23 / 25	sagarparasher04@gmail.com
samarthmadaan72@gmail.com	23 / 25	samarthmadaan72@gmail.com
yuvsharma464@gmail.com	24 / 25	yuvsharma464@gmail.com
ajeetsingh4656@gmail.com	24 / 25	ajeetsingh4656@gmail.com
shreymitra376@gamil.com	20 / 25	shreymitra376@gmail.com
nsnikitasoni17@gmail.com	17 / 25	nsnikitasoni17@gmail.com
namitjoshi2004@gmail.com	24 / 25	namitjoshi2004@gmail.com

Name	Roll No.	1) What built-in list method would you use to remove items from a list?
Priyanjal Malhotra	13	.pop() method
Gopal Kumar	02721302022	.pop() method
Yatharth singhal	04321302022	.pop() method
Siddharth popli	53	.pop() method
Rohit Tanwar	00121302022	.pop() method
Vrinda Suneja	03521302022	.pop() method
Prajjwal Dwivedi	2.10132E+12	pop(my_list)
vivek	2200041	pop(my_list)
Keshav Kumar	2200040	.pop() method

Harsh gour	2.10132E+12	.pop() method
DEEPAK	2.10132E+12	.pop() method
Gibrail Zaidi	2.10132E+12	.pop() method
Gazab Bhati	2.10132E+12	.pop() method
Aditya Mishra	2.10132E+12	.pop() method
Fazal Singh	2.10132E+12	.pop() method
ANUJ MISHRA	2.10132E+12	.pop() method
Adarsh Kumar	2.10132E+12	pop(my_list)
sagar sharma	2200122	del(my_list)
Anas choudhary	2200094	.pop() method

Harsh Sharma	2.10132E+12	.pop() method
faizan nasim	2200067	.delete() method
Md Amber Khan	2.10132E+12	.pop() method
Ayush kumar Srivastav	2.10132E+12	.pop() method
Gaganshu yadav	2.10132E+12	.pop() method
Anzar Hashmat	2200095	.pop() method
Arun Pandey	2.10132E+12	.pop() method
Anshuman Soni	2.10132E+12	pop(my_list)
himanshu sharma	2200071	.pop() method
Vishesh Verma	2.10132E+12	.pop() method

Kamran Ahmad	2.19132E+12	.pop() method
Viraz Anand Gupta	2.10132E+12	.pop() method
manish kumar singh	2.10132E+12	pop(my_list)
shivam singh	2.10132E+12	.pop() method
Himanshu Singh Rautela	2.10132E+12	.pop() method
Sumit rawat	2.10132E+12	.pop() method
Himanshu singh	2.10132E+12	.pop() method
sourav kumar	2.10132E+12	pop(my_list)
Nav Verma	01217002022	.pop() method
Naveen Sangwan	02417002022	.pop() method

VAIBHAV JAIN	00617002022	pop(my_list)
Fagun	41	.pop() method
Dev Mohan Sharma	01017002022	.pop() method
Sidra Tabassum	04617002022	.delete() method
Jhanvi Khanna	35617002022	del(my_list)
Aviral Rastogi	01817002022	.pop() method
Aanchal	04717002022	.pop() method
Priya kumari	00217002022	del(my_list)
Dff	F	.delete() method
Shankar	3	pop(my_list)

Parth Goyal	00421302022	pop(my_list)
Manya Mittal	35321302022	.pop() method
Rukaiya	38	.pop() method
Sagar parasher	22	.pop() method
Samarth nadaan	01421302022	.delete() method
Yuv Sharma	32	.pop() method
Ajeet Singh	02121302022	.pop() method
Shrey	17	pop(my_list)
Nikita Soni	02321302022	pop(my_list)
Namit Joshi	8	.pop() method

Mankirat Singh	06	.delete() method

2) What is the purpose of an if/else statement?	3) What does the built-in map() function do?	4) If you don't explicitly return a value from a function, what happens?
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It creates a mapping between two different elements of different iterables.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It converts a complex value type into simpler value types.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It creates a mapping between two different elements of different iterables.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It runs one chunk of code if all the imports were successful, and another chunk of code if the imports were not successful.	It applies a function to each item in an iterable and returns the value of that function.	The function will enter an infinite loop because it won't know when to stop executing its code.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.

It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	The function will return a RuntimeError if you don't return a value.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
chunk of code to run if the is enough memory to handle it, and which chunk of code to run if there is not enough memory to handle it.	It creates a mapping between two different elements of different iterables.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It creates a mapping between two different elements of different iterables.	If the return keyword is absent, the function will return True.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It creates a mapping between two different elements of different iterables.	If the return keyword is absent, the function will return True.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return True.

It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return True.
It runs one chunk of code if all the imports were successful, and another chunk of code if the imports were not successful.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return True.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return True.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	The function will return a RuntimeError if you don't return a value.
It tells the computer which chunk of code to run if the instructions you coded are incorrect.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return True.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	The function will return a RuntimeError if you don't return a value.
chunk of code to run if the is enough memory to handle it, and which chunk of code to run if there is not enough memory to handle it.	It creates a mapping between two different elements of different iterables.	If the return keyword is absent, the function will return True.
It runs one chunk of code if all the imports were successful, and another chunk of code if the imports were not successful.	It converts a complex value type into simpler value types.	If the return keyword is absent, the function will return True.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	The function will return a RuntimeError if you don't return a value.

It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	The function will return a RuntimeError if you don't return a value.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It runs one chunk of code if all the imports were successful, and another chunk of code if the imports were not successful.	It creates a path from multiple values in an iterable to a single value.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It runs one chunk of code if all the imports were successful, and another chunk of code if the imports were not successful.	It converts a complex value type into simpler value types.	The function will enter an infinite loop because it won't know when to stop executing its code.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It creates a path from multiple values in an iterable to a single value.	If the return keyword is absent, the function will return True.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.		If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.		If the return keyword is absent, the function will return None.

It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It creates a path from multiple values in an iterable to a single value.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It creates a mapping between two different elements of different iterables.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It converts a complex value type into simpler value types.	If the return keyword is absent, the function will return None.
		The function will enter an infinite loop because it won't know when to stop executing its code.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It creates a mapping between two different elements of different iterables.	The function will return a RuntimeError if you don't return a value.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	The function will enter an infinite loop because it won't know when to stop executing its code.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.		If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It converts a complex value type into simpler value types.	If the return keyword is absent, the function will return True.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.

It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It converts a complex value type into simpler value types.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It creates a path from multiple values in an iterable to a single value.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.
It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It applies a function to each item in an iterable and returns the value of that function.	If the return keyword is absent, the function will return None.

It executes one chunk of code if a condition is true, but a different chunk of code if the condition is false.	It converts a complex value type into simpler value types.	If the return keyword is absent, the function will return True.

5) When does a for loop stop iterating?	6) What is key difference between a set and a list?	7) Review the code below. What is the correct syntax for changing the price to 1.5? fruit_info = { 'fruit': 'apple',
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an if/else statement that contains a break keyword	A set is an ordered collection of non-unique items. A list is an unordered collection of unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an infinite loop	A set is an ordered collection unique items. A list is an unordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5

when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an if/else statement that contains a break keyword	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an infinite loop	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an infinite loop	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an if/else statement that contains a break keyword	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	my_list['price'] == 1.5

when it encounters an if/else statement that contains a break keyword	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an if/else statement that contains a break keyword	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an if/else statement that contains a break keyword	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	my_list['price'] == 1.5
when it encounters an if/else statement that contains a break keyword	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	my_list['price'] == 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5

when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an if/else statement that contains a break keyword	A set is an ordered collection unique items. A list is an unordered collection of non-unique items.	my_list['price'] == 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5

when it encounters an if/else statement that contains a break keyword	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an if/else statement that contains a break keyword	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an ordered collection unique items. A list is an unordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an ordered collection unique items. A list is an unordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when the runtime for the loop exceeds O(n^2)	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	my_list['price'] == 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5

when it encounters an if/else statement that contains a break keyword	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it encounters an infinite loop	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an ordered collection of non-unique items. A list is an unordered collection of unique items.	fruit_info ['price'] = 1.5
when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an unordered collection unique items. A list is an ordered collection of non-unique items.	fruit_info ['price'] = 1.5

when it has assessed each item in the iterable it is working on or a break keyword is encountered	A set is an ordered collection unique items. A list is an unordered collection of non-unique items.	fruit_info ['price'] = 1.5

8) You are given a piece of code. Assume m and n are already defined as some positive integer value. When it completes, how many tuples	9) Assume m, n and p are positive integers. In the following comprehension, how many times will the function randint be called?	10) What is the correct syntax for replacing the string apple in the list with the string orange?
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m + n + p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m + n + p	my_list[1] = 'orange'
m * n	the greater value of (m,n,p)	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'

m * n		my_list[1] = 'orange'
m * n	m + n + p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n		my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'

m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	1 million	my_list[1] = 'orange'
m + n	the greater value of (m,n,p)	my_list[1] = 'orange'
m * n	m + n + p	my_list[1] = 'orange'

m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m + n + p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m + n	the greater value of (m,n,p)	orange = my_list[1]
m	m _ n _ p	my_list[1] = 'orange'
m	m _ n _ p	my_list[1] = 'orange'

m * n	m + n + p	my_list[1] = 'orange'
m + n	m + n + p	my_list[1] = 'orange'
m + n	m _ n _ p	my_list[1] = 'orange'
m + n	m _ n _ p	my_list[1] = 'orange'
m + n	the greater value of (m,n,p)	my_list[1] = 'orange'
m * n	m + n + p	my_list[1] = 'orange'
m + n	m + n + p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m	the greater value of (m,n,p)	my_list[1] == orange
m * n	m _ n _ p	my_list[1] = 'orange'

m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m + n + p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'
m * n	m _ n _ p	my_list[1] = 'orange'

m * n		my_list[1] = 'orange'

11) Which comparison of lists and tuples in Python is correct?	12) What will this code output to the screen?
	for i in range(5):
	  print(i)
	else:
		13) Which collection is ordered, changeable, and allows duplicate members?

Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use lists instead of tuples when you have a collection of related but dissimilar objects.	You will get a syntax error.	SET
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use lists instead of tuples when you have a collection of related but dissimilar objects.	0 1 2 3 4 Done!	LIST
Use lists instead of tuples when you have a collection of related but dissimilar objects.	0 1 2 3 4 Done!	LIST
Use lists instead of tuples when the position of elements is important.	0 1 2 3 4 5 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST

Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 5 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST

Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use lists instead of tuples when you have a collection of related but dissimilar objects.	0 1 2 3 4 Done!	LIST
Use lists instead of tuples when you have a collection of related but dissimilar objects.	0 1 2 3 4 Done!	LIST

Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use lists instead of tuples when you have a collection of related but dissimilar objects.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists when you have a common collection of similar objects.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists when you have a common collection of similar objects.	1 2 3 4 5 Done!	TUPLE
	0 1 2 3 4 Done!	LIST
	0 1 2 3 4 Done!	LIST

Use lists instead of tuples when the position of elements is important.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists when you have a common collection of similar objects.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists when you have a common collection of similar objects.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists when you have a common collection of similar objects.	0 1 2 3 4 Done!	TUPLE
Use lists instead of tuples when you have a collection of related but dissimilar objects.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists when you have a common collection of similar objects.	You will get a syntax error.	TUPLE
Use tuples instead of lists when you have a common collection of similar objects.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists when you have a common collection of similar objects.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 5 Done!	DICTIONARY
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST

Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use lists instead of tuples when you have a collection of related but dissimilar objects.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST
Use tuples instead of lists for functions that need to return multiple values.	0 1 2 3 4 Done!	LIST

	0 1 2 3 4 5 Done!	DICTIONARY

14) What happens if the file is not found in the following Python code? a=False **while not** a:	15) What will be the output of the following Python code? lst = [1, 2, 3] lst[3]	16) Identify the type of error in the following Python codes? Print(“Good Morning”) **print**(“Good night)
Input output error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
Input output error	IndexError	Syntax, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Syntax, Syntax
Input output error	IndexError	Syntax, Syntax
No error	ValueError	Semantic, Syntax
Input output error	IndexError	Syntax, Syntax
No error	IndexError	Semantic, Syntax

Input output error	IndexError	Semantic, Syntax
Input output error	IndexError	Semantic, Syntax
Input output error	IndexError	Syntax, Semantic
Input output error	IndexError	Syntax, Semantic
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
Input output error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax

No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
Input output error	IndexError	Semantic, Syntax
Input output error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Syntax, Semantic
Name error	IndexError	Semantic, Syntax
Input output error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax

No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
Input output error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
Input output error	IndexError	Semantic, Syntax
No error	IndexError	Syntax, Syntax
No error	IndexError	Syntax, Syntax

Assertion error	IndexError	Semantic, Syntax
	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
	IndexError	Semantic, Syntax
Input output error	ValueError	Semantic, Syntax
Assertion error	IndexError	Semantic, Semantic
No error	IndexError	Semantic, Syntax
Input output error	IndexError	Semantic, Syntax
Assertion error	ValueError	Semantic, Semantic
No error	IndexError	Semantic, Syntax

No error	IndexError	Semantic, Syntax
No error	IndexError	Syntax, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	IndexError	Semantic, Syntax
No error	ValueError	Semantic, Syntax
No error	IndexError	Semantic, Syntax

Name error	IndexError	Syntax, Syntax

17) An exception is _____	18) How many keyword arguments can be passed to a function in a single function call?	19) How many except statements can a try-except block have?
an object	zero or more	more than zero
an object	zero or more	more than zero
a module	one or more	more than one
an object	zero or more	more than zero
an object	zero or more	more than one
an object	zero or more	more than one
a special function	one	more than one
an object	one or more	more than one
an object	zero or more	more than zero

an object	zero or more	more than zero
an object	one or more	more than one
an object	one or more	more than one
an object	one or more	more than one
an object	zero or more	more than zero
an object	zero or more	more than zero
an object	zero or more	more than zero
an object	zero	1
an object	zero	1
an object	one or more	more than one

an object	zero or more	more than zero
an object	zero or more	more than zero
an object	zero	1
an object	zero	1
an object	one	more than one
an object	zero or more	more than zero
an object	one or more	1
an object	zero or more	1
an object	zero or more	1
an object	zero	1

an object	zero	1
an object	zero	1
an object	zero	more than one
an object	one	more than one
an object	zero or more	more than zero
an object	zero or more	more than zero
an object	zero or more	more than one
a special function	zero or more	1
a standard module	zero or more	1
a special function	zero or more	1

an object	one or more	more than zero
a special function	zero or more	
an object	zero or more	more than one
a standard module	zero or more	more than one
an object	one	1
a special function	zero or more	1
an object	zero	more than zero
an object	zero or more	more than one
a standard module	zero or more	more than one
an object	zero or more	more than zero

an object	zero or more	more than zero
an object	zero or more	more than one
an object	zero or more	more than zero
an object	zero or more	more than zero
an object	zero or more	more than zero
an object	zero or more	more than zero
an object	zero or more	more than zero
a special function	one	more than one
a module	zero or more	1
an object	zero or more	more than zero

a special function	one or more	more than one

20) Which of the following will print the pi value defined in math module?	21) Which operator is used in Python to import modules from packages?	22) How is a function declared in Python?
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
print(math.pi)	.	
from math import pi print(pi)	.	def function_name():
print(math.pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	*	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():

from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
print(math.pi)	->	def function_name():
print(math.pi)	->	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
print(pi)	.	def function_name():
print(pi)	->	
from math import pi print(pi)	.	def function_name():

from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
print(pi)	.	def function_name():
print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
print(math.pi)	.	def function_name():

print(pi)	.	def function_name():
print(pi)	.	def function_name():
print(pi)	.	def function_name():
print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	*	def function_name():
from math import pi print(math.pi)	*	def function_name():
from math import pi print(math.pi)	*	def function_name():

from math import pi print(math.pi)	.	def function_name():
from math import pi print(math.pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(math.pi)	->	def function_name():
from math import pi print(math.pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	*	declare function function_name():
from math import pi print(pi)	.	def function_name():

from math import pi print(pi)	.	def function_name():
print(math.pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():
from math import pi print(pi)	.	def function_name():

from math import pi print(pi)	.	def function_name():

23) Which one of the following is the correct way of calling a function?	24) Choose the correct option with reference to below Python code? def fn(a):	25) Which one of the following is incorrect?
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the formal argument.	In order to change the value of global variable inside function, keyword global is used.
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	In order to change the value of global variable inside function, keyword global is used.
ret function_name()	a is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space

function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the formal argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the formal argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space

function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
ret function_name()	fn(x) is the function signature.	In order to change the value of global variable inside function, keyword global is used.
function_name()	x is the formal argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	fn(x) is the function signature.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
ret function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space

function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
ret function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the formal argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	a is the actual argument.	The variables used outside function are called global variables
function_name()	a is the actual argument.	The variables used outside function are called global variables

function_name()	x is the formal argument.	In order to change the value of global variable inside function, keyword global is used.
function_name()	x is the actual argument.	The variables used inside function are called local variables.
function_name()	fn(x) is the function signature.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function function_name()	x is the actual argument.	In order to change the value of global variable inside function, keyword global is used.
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the formal argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
call function_name()	x is the formal argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space

function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function function_name()	x is the formal argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space
function_name()		In order to change the value of global variable inside function, keyword global is used.
function_name()	x is the actual argument.	The local variables of a particular function can be used inside other functions, but these cannot be used in global space

function_name()	x is the formal argument.	The variables used inside function are called local variables.