

1. Given the following variables, write the first three lines that the sample programs will output. If there is an error, write "error".

```
a = [2, 1, 0]
b = "2,1,0"
c = {0:1, 1:3, 2:5}
```

for i in a: print(i)	
for i in b: print(i)	
for i in c: print(i)	

for i in range(len(a)): print(i)	
for i in range(len(b)): print(i)	
for i in range(len(c)): print(i)	

for i in a: print(c[i])	
for i in b: print(b[i])	
for i in c: print(a[i])	

for i in range(len(a)): print(a[i])	
for i in range(len(c)): print(c[i])	
for i in c: print(i[0]) print(i[1])	

2. The following questions refer to this list:

```
[100, [5,10,[1,2]], {"1":"orangejuice", "2":"applesauce"}, "WONDERWOMAN"]
```

For each of the questions below, write an indexing statement to isolate the value in question. If it's not possible, write "N/A".

As an example, if you were asked to isolate the value 100 from the list you would write [0] as your answer.

You are only writing an indexing statement – you do not need to include a variable name.

2.1	Isolate the integer 5	
2.2	Isolate the string "2"	
2.3	Isolate the integer 1	
2.4	Isolate the string "WONDERWOMAN"	
2.5	Isolate the string "sauce"	

3. The program below ("Program to Fix") is missing critical pieces of code to make it function as expected. The program should open a data file named 'data.txt' which contains a series of names and their personality types, organized as follows:

**data.txt**

Harry, ISFP, Leo  
Ron, ESFP, Pisces  
Hermione, ESTJ, Virgo

The program should create a dictionary of wizards as such, with names as the keys and a list of personality traits as the values:

```
wizards = {  
    "Harry": ["ISFP", "Leo"],  
    "Ron": ["ESFP", "Pisces"],  
    "Hermione": ["ESTJ", "Virgo"]  
}
```

**Program to Fix**

```
fp = 3.1('data.txt', "3.2")  
data = fp.3.3()  
data = data.3.4(" 3.5 ")  
  
wizards = 3.6  
  
for d in 3.7:  
    p = d.3.8(" 3.9 ")  
    wizards[3.10] = [3.11, 3.12]
```

Options to Choose From:

(A) read	(M) r
(B) open	(N) w
(C) close	(O) a
(D) trim	(P) [0]
(E) split	(Q) [1]
(F) join	(R) [2]
(G) " "	(S) range(len(data))
(H) "\n"	(T) data
(I) "\t"	(U) p[0]
(J) ", "	(V) p[1]
(K) {}	(W) p[2]
(L) []	(X) d

Your Answers (write the letter):

3.1	
3.2	
3.3	
3.4	
3.5	
3.6	
3.7	
3.8	
3.9	
3.10	
3.11	
3.12	

4. Trace the output of the following programs. These programs all refer to the three classes listed below. If a program crashes briefly explain why. Write your answers in the boxes provided.

```
class A:
    def __init__(self):
        self.x = 0
        self.y = 0
        self.z = 0
        print ("A!")
    def f1(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
        return True

class B:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
        print ("B!")
    def f1(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
        return True
    def f2(self):
        return self.x + self.y + self.z
```

```
class C:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
        print ("C!", self.f2() )
    def f1(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
        return True
    def f2(self):
        return self.x + self.y + self.z
    def f3(self, other):
        return self.f2() + other.f2()
```

```
q = A()
r = A()
print (q.x, q.y, q.z);
print (r.x, r.y, r.z);
q.f1(9,1,3)
print (q.x, q.y, q.z);
print (r.x, r.y, r.z);
```

```
q = A()
r = B(1,2,3)
print (q.x, q.y, q.z);
print (r.x, r.y, r.z);
q.f1(3,2,1)
r.f1(6,5,4)
print (q.x, q.y, q.z);
print (r.x, r.y, r.z);
```

```
q = A()
r = B(0,0,0)
s = C(0,0,0)
print (q == r, q == s, r == s)
print (q.x == r.x, q.x == s.x, r.x == s.x)
print (q.f2(), r.f2(), s.f2())
q = B(0,0,0)
r = B(1,1,1)
```

```
s = C(3,3,3)
print (s.f3(q), s.f3(r))
q.f1(1,1,1)
print (s.f3(q), s.f3(r))
```

5. Write a FUNCTION that operates as follows:

NAME: lists\_to\_dict

INPUT: two lists

PROCESSING: The two lists being provided are assumed to be “parallel” lists of the same length. Your function should create a dictionary using the first list as its keys and the second list as its values. For example, consider if the function is called with the following lists:

```
["Spirited Away", "Princess Mononoke", "Howl's Moving Castle"]
[2001, 1997, 2004]
```

Your function should return the following dictionary:

```
{"Spirited Away": 2001,
 "Princess Mononoke": 1997,
 "Howl's Moving Castle": 2004}
```

OUTPUT: a dictionary

6. Write a FUNCTION that operates as follows:

NAME: get\_key\_by\_value

INPUT: [1] a value to search for (any data type) and [2] a dictionary

PROCESSING: Your function should find all keys in the dictionary that are holding the supplied value.

For example, consider the following dictionary:

```
dictionary = {"a":5, "b":10, "c":15, "d":3, "e":3, "f":3 }
```

Calling your function should return the following lists:

```
key1 = get_key_by_value(5, dictionary)
print (key1) # ['a']
key2 = get_key_by_value(3, dictionary)
print (key2) # ['e', 'f', 'd']
key3 = get_key_by_value(9, dictionary)
print (key3) # []
```

7. Write a program that prompts the user to enter a series of test scores within the range of 0 and 100 (inclusive) as integers. The user will enter all scores on a single line, and scores will be separated by spaces. You cannot assume the user will supply valid integers, nor can you assume that they will enter positive numbers, so make sure that you validate your data. Your program should not crash under any circumstances, and you do not need to re-prompt the user once they have entered a single line of data.

Once you've collected the scores you should generate the following statistics:

- The scores printed in ascending order
- Average of all scores
- Total # of scores
- The highest score entered
- The lowest score entered
- The number of times the highest score was entered
- The number of times the lowest score was entered

Here's a sample running of your program (the shaded line represents user input):

```
Enter a series of test scores separated by spaces: 99 105 -10 30 12 30 12 12 apple pear
twelve 12 30
* 99: valid score
* 105: scores above 100 not allowed
* -10: negative scores not allowed
* 30: valid score
* 12: valid score
* 30: valid score
* 12: valid score
* 12: valid score
* apple: not a valid score!
* pear: not a valid score!
* twelve: not a valid score!
* 12: valid score
* 30: valid score
Scores in ascending order: [12, 12, 12, 12, 30, 30, 30, 99]
Average score: 29.625
Highest score: 99
Lowest score: 12
Number of times the highest score occurs: 1
Number of times the lowest score occurs: 4
```

**[Write answer on next page]**





## Python Core Language Elements & Functions

and	int	<u>Module Functions</u>	<u>String Methods</u>
chr	max	random.randint	split()
def	min		find()
del	not	<u>File Methods</u>	isalpha()
elif	open	close()	isdigit()
else	or	read()	islower()
except	ord	write()	isupper()
float	print		isspace()
for	range	<u>List Methods</u>	isalnum()
format	return	append()	lower()
global	str	index()	upper()
if	str.lower	insert()	
import	str.upper	remove()	<u>Dictionary Methods</u>
in	try	reverse()	clear()
input	while	sort()	keys()
			values()
			items()
			get()

## ASCII Code Table

0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Decimal ASCII Chart