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)`
	`2`
	`1`
	`0`

`for i in b:`
`  print(i)`
	`2`
	`,`
	`1`

`for i in c:`
`  print(i)`
	`0`
	`1`
	`2`

`for i in range(len(a)):`
`  print(i)`
	`0`
	`1`
	`2`

`for i in range(len(b)):`
`  print(i)`
	`0`
	`1`
	`2`

`for i in range(len(c)):`
`  print(i)`
	`0`
	`1`
	`2`

`for i in a:`
`  print(c[i])`
	`5`
	`3`
	`1`

`for i in b:` `  print(b[i])`	`error`

`for i in c:`
`  print(a[i])`
	`2`
	`1`
	`0`

`for i in range(len(a)):`
`  print(a[i])`
	`2`
	`1`
	`0`

`for i in range(len(c)):`
`  print(c[i])`
	`1`
	`3`
	`5`

`for i in c:`
`  print(i[0])`
`  print(i[1])`
	`error`

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	`[1][0]`
2.2	Isolate the string "2"	N/A
2.3	Isolate the integer 1	`[1][2][0]`
2.4	Isolate the string "WONDERWOMAN"	`[3]`
2.5	Isolate the string "sauce"	`[2]["2"][5:]`

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

   Answers:

3.1	B	open
3.2	M	r
3.3	A	read
3.4	E	split
3.5	H	\n
3.6	K	{}
3.7	T	data
3.8	E	split
3.9	J	,
3.10	U	p[0]
3.11	V	p[1]
3.12	W	p[2]

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);
```

```
A!
A!
0 0 0
0 0 0
9 1 3
0 0 0
```

```
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);
```

```
A!
B!
0 0 0
1 2 3
3 2 1
6 5 4
```

```
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)
```

```
A!
B!
C! 0
False False False
True True True
ERROR: q.f2()→ A objects don’t have f2
function
```

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

```
C! 9
ERROR: q is not defined
```

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

```
def lists_to_dict(list1, list2):
    d = {}
    for i in range(len(list1)):
        d[list1[i]] = list2[i]
    return d
```

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) # ['d', 'e', 'f']
key3 = get_key_by_value(9, dictionary)
print (key3) # []
```

```
def get_key_by_value(value, d):
    keys = []
    for i in d.items(): # i[0] is the key, i[1] is the value
        if i[1] == value:
            if i[0] not in keys:
                keys.append(i[0])
    return keys
```

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
```

```
data = input("Enter a series of test scores sperated by spaces: ")
data_list = data.split(" ")

scores = [] # to store valid scores
for d in data_list:
    try:
        d = int(d) # try to convert to int
    except:
        # if int cannot be converted
        response = "not a valid score!"
    else:
        if d > 100:
            response = "scores above 100 not allowed"
        elif d < 0:
            response = "negative scores not allowed"
        else:
            response = "valid score"
            scores.append(d) # add valid score to scores list
    print("*", str(d) + ":", response)

scores.sort()
print("Scores in ascending order:", scores)
print("Average score:", sum(scores)/len(scores))
print("Highest score:", max(scores))
print("Lowest score:", min(scores))
print("Highest score occurances:", scores.count(max(scores)))
print("Lowest score occurances:", scores.count(min(scores)))
```