Evaluation of Explanation for Fault Localization (Part 1)

In this section, you will assess the explanations provided regarding why a specific line of code is deemed faulty. There will be two explanations given for each faulty code. The assessment would focus on the correctness, clearness, and informativeness of the explanation.

The faulty code is extracted from a Python student assignment related to the following task description:
The `unique_day` function takes a day (provided as a string) and a list of possible_birthdays (represented as tuples containing month and day) as input. It checks if a given day is unique within the list of possible birthdays.

The `unique_month` function takes a month (provided as a string) and a list of possible_birthdays (represented as tuples containing month and day) as input. It checks if a given month is unique within the list of possible birthdays.

The `contains_unique_day` function takes a month (provided as a string) and a list of possible_birthdays (represented as tuples containing month and day) as input. It checks if there exists a birthday with a unique day within the list for a specific month.

Faulty Code 1

```
01  def unique_month(month, possible_birthdays):
02
03      month = 0 #This is the faulty line!
04
05      for i in range(len(possible_birthdays)):
06          if possible_birthdays[i][0] == month:
07              day = day + 1
08
09      if days == 1:
10          return True
11      else:
12          return False
```

Explanation 1:

This line is assigning `0` to variable `month`. However, `month` is the input variable. This line overwrite the previous assign value to 0. This could potentially impact the result, causing it to become incorrect.

6. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

7. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

8. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
The line initializes the variable 'month' to 0, which is unnecessary and incorrect since the input parameter 'month' is already provided. This line should have been removed or renamed to avoid confusion.

9. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

10. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

11. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 2

```
1  def contains_unique_day(month, possible_birthdays):
2      return #This is the faulty line!
```

Explanation 1:
The return statement in the `contains_unique_day` function is returning `None` instead of the expected boolean value.

12. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

13. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

14. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
This line contains a return statement that returns nothing, resulting in the function always returning None when executed. This implementation of the contains_unique_day function is incomplete.

15. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

16. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

17. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 3

```
01  def count_dates(date, possible_birthdays):
02      count = 0
03      for i in possible_birthdays:
04          if i[1] == date:
05              count += 1
06      return count
07
08  def unique_day(date, possible_birthdays):
09      if count_dates(date, possible_birthdays) == 1:
10          return True
11      else:
12          return False
13
14  def count_months(month, possible_birthdays):
15      count = 0
16      for i in possible_birthdays:
17          if i[0] == month:
18              count += 1
19      return count
20
21  def unique_month(month, possible_birthdays):
22      if count_months(month, possible_birthdays) == 1:
23          return True
24      else:
25          return False
26
27
28  def contains_unique_day(month, possible_birthdays):
29      days_in_month = ()
30      for i in possible_birthdays:
31          if i[0] == month:
32              days_in_month += (i[1],)
33      for i in range(len(days(month, possible_birthdays))): #This is the faulty line!
34          if unique_day(i, possible_birthdays):
35              return True
36          else:
37              return False
```

Explanation 1:

This line tries to iterate over the range of the length of `days(month, possible_birthdays)`. However, the function `days()` is not defined in the code, which causes a `NameError`. It seems like this line is referencing a non-existent function and needs to be revised.

18. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

19. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

20. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
The line utilize `days()` function for the iteration. However, `days()` is never defined in the code. This can raise an error.

21. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

22. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

23. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 4

```
01  def unique_day(day, possible_birthdays):
02      occur = 0
03      for i in possible_birthdays:
04              if day == i[1]:
05                  occur += 1
06      if occur == 1:
07          return True
08      else:
09          return False
10
11  def unique_month(month, possible_birthdays):
12      occur = 0
13      for i in possible_birthdays:
14              if month == i[0]:
15                  occur += 1
16      if occur == 1:
17          return True
18      else:
19          return False
20
21
22  def contains_unique_day(month, possible_birthdays):
23      occur = 0
24      for i in possible_birthdays:
25              if month == i[1]: #This is the faulty line!
26                  occur += 1
27      if occur == 1:
28          return True
29      else:
30          return False
```

Explanation 1:
This line only check whether the month is equal with the i[1]. However, i[1] is represent day and i[0] is represent month. This line is incorrect as it compare the month with i[1] (day). Furthermore, it also does not check whether the current day is unique in possible_birthdays.

24. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

25. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

26. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
This line checks if the given month matches the second element (day) of the current tuple `i`. This line should compare the month with the first element (month) of the tuple 'i' instead of the second element.

27. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

28. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

29. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Evaluation of Explanation for Fault Localization (Part 2)

In this section, you will assess the explanations provided regarding why a specific line of code is deemed faulty. There will be two explanations given for each faulty code. The assessment would focus on the correctness, clearness, and informativeness of the explanation.

The faulty code is extracted from a Python student assignment related to the following task description: **The function `remove_extras` takes in a list and returns a new list with all repeated occurrences of any element removed. This new list should maintain the same order as the input list. For example, remove_extras([5, 2, 1, 2, 3]) returns the list [5, 2, 1, 3].**

Faulty Code 5

```
1  def remove_extras(lst):
2      result = []
3      for i in lst:
4          if i not in result:
5              result += result + i #This is the faulty line!
6      return result
```

Explanation 1:
The line is concatenating the 'result' list with itself and the current element 'i'. However, the concatenation operation is incorrect, resulting in a TypeError. It should have been 'result.append(i)' to add the current element 'i' to the 'result' list.

30. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

31. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

32. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
This line attempts to add the `result` list with the `result` itself and `i` which is the element from `lst`. However, adding `i` to the `result` using `+` can raise TypeError as it is concatenating list with integer. Furthermore, it would produce incorrect results as the element `result` would be duplicate.

33. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

34. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

35. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 6

```
01  def remove_extras(lst):
02      new = []
03      for x in lst:
04          if lst.count(x) > 1: #This is the faulty line!
05              new += []
06          else:
07              new = new.append(x)
08      return new
09      pass
```

Explanation 1:
If there are more than 1 occurrences of x in the `lst`, it appends nothing to the list indicated by `[]`. However, by doing this, all the elements that show up more than once will not included in the `new` list. In the context of removing extras, the desired outcome is to retain elements that are duplicates while eliminating the extras. Therefore, this line can lead to unintended results.

36. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

37. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

38. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
The line checks if the count of 'x' in 'lst' is greater than 1. This condition determines if an element is repeated in the list.

39. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

40. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

41. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 7

```
1  def remove_extras(lst):
2      seq = (lst[0],)
3      for i in lst:
4          if i not in seq:
5              seq = seq + (lst[i],) #This is the faulty line!
6      return seq
7      pass
```

Explanation 1:
This line tries to add `lst[i]` to `seq`. However, `i` is already the element and not an index, so it can generate an error.

42. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

43. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

44. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
The code attempts to add the current element lst[i] to the seq tuple, but it should use i directly as the element itself.

45. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

46. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

47. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 8

```
1  def remove_extras(lst):
2      one = [lst[0],] #This is the faulty line!
3      for repeat in lst:
4          if repeat not in one:
5              one += [repeat,]
6      return one
```

Explanation 1:
The initial value of `one` is set to contain the first element of the `lst`. However, this line will throw an IndexError if `lst` is empty.

48. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

49. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

50. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
The code attempts to initialize a new list 'one' with the first element of the input list 'lst'. However, if the input list is empty, an IndexError occurs because there is no element at index 0.

51. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

52. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

53. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 9

```
01  def remove_extras(lst):
02      new_lst = []
03      for i in lst:
04          if i not in new_lst:
05              new_lst = new_lst + i #This is the faulty line!
06          else:
07              continue
08      return new_lst
```

Explanation 1:
The line attempts to concatenate a list with an integer, resulting in a TypeError. To fix this, we should use the append() method to add the element 'i' to 'new_lst' instead of concatenating it.

54. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

55. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

56. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
This line attempts to add "i" element on the "new_lst". However, when adding "i" to the "new_lst" using "+" it can generate TypeError as it is concatenating type list with integer.

57. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

58. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

59. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 10

```
01  def remove_extras(lst):
02      l=len(lst)
03      for i in l: #This is the faulty line!
04          for j in range(i+1,l):
05              if lst[i]==lst[j]:
06                  del lst[j]
07      return lst
08
09      # your code here
10      pass
```

Explanation 1:

This line is designated for an iteration. In this for loop, it used "l" while "l" is an integer generated in the previous line which is length of "lst" array. This is incorrect way of loop, as the "l" that used in this line is integer, it should use "range" for the loop.

60. Is the explanation correct? *

Mark only one oval.

- Yes
- No

61. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

62. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
The loop variable `i` is assigned the value of `l`, which is an integer and not an iterable object. To fix this, we need to change the loop statement to iterate over a range of values up to `l`, instead of iterating over `l` itself.

63. Is the explanation correct? *

Mark only one oval.

- Yes
- No

64. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

65. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Evaluation of Explanation for Fault Localization (Part 3)

In this section, you will assess the explanations provided regarding why a specific line of code is deemed faulty. There will be two explanations given for each faulty code. The assessment would focus on the correctness, clearness, and informativeness of the explanation.

The faulty code is extracted from a Python student assignment related to the following task description:
Given a list of people that is represented using a tuple (<gender>, <age>), function `sort_age` sorts the people and return a list in an order such that the older people are at the front of the list. An example of the list of people is [("M", 23), ("F", 19), ("M", 30)]. The sorted list would look like [("M", 30), ("M", 23), ("F", 19)]. You may assume that no two members of the list of people are of the same age.

Faulty Code 11

```
01  def sort_age(lst):
02      """selection sort"""
03      l = len(lst)
04      for i in range(l):
05          largest = lst[i]
06          for j in range(i+1,l):
07              if lst[j][1] > largest[1]:
08                  largest = lst[j] #assign new largest value
09                  lst[i],lst[j] = lst[j],lst[j] #swap positions if larger #This is the faulty line!
10      return lst
```

Explanation 1:
The line is intended to swap the positions of the current element with the larger element. However, it mistakenly swaps lst[j] with itself (lst[j],lst[j]). The correct swap should be lst[i], lst[j] = lst[j], lst[i].

66. Is the explanation correct? *

Mark only one oval.

- Yes
- No

67. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

68. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
This line is supposed to assign the original value of lst[i] to lst[j] to swap their positions. However it is wrongly assigning the original value of lst[j] back to lst[j].

69. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

70. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

71. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 12

```
1  def sort_age(lst):
2      return lst.sort(key=lambda x:x[1],reverse=True) #This is the faulty line!
```

Explanation 1:
The line returns None instead of the sorted list because the list.sort() method sorts the list in-place and returns None. Therefore, we need to sort the list using lst.sort() and then return lst.

72. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

73. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

74. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
The line uses `lst.sort()` to sort the original `lst`. However, `lst.sort()` does not provide a return value, so the return value will always be None.

75. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

76. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

77. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 13

```
01  def sort_age(lst):
02      new_lst = []
03      while lst:
04          oldest = lst [0]
05          for i in range(len(lst)):
06              if lst[i][1] > oldest[1]:
07                  oldest = lst[i]
08          lst.remove(oldest)
09          new_lst.append(oldest)
10      print(new_lst) #This is the faulty line!
```

Explanation 1:

The line should be modified to return `new_lst` instead of printing it.

78. Is the explanation correct? *

Mark only one oval.

- Yes
- No

79. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

80. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
This line is printing the new_lst list. However, this function is missing a return type since `print()` does not return any value.

81. Is the explanation correct? *

Mark only one oval.

- Yes
- No

82. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

83. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 14

```
01  def sort_age(lst):
02      holder=[]
03      if lst==[]:
04          return []
05      for x in lst:
06          if holder==[]:
07              holder=x
08          elif x[1]>holder[1]:
09              holder=x
10      return holder+sort_age(lst[1:]) #This is the faulty line!
```

Explanation 1:

Concatenating a tuple with the result of the recursive call to sort_age may cause a TypeError.

84. Is the explanation correct? *

Mark only one oval.

- Yes
- No

85. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

86. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:

This line makes a recursive call without a reduction step to remove `holder` from `lst`. This will result in an infinite loop.

87. Is the explanation correct? *

Mark only one oval.

- Yes
- No

88. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

89. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 15

```
01  def sort_age(lst):
02      sort = []
03      while lst:
04          oldest = lst[0]
05          for i in lst:
06              if i[1] > oldest[1]:
07                  oldest = i
08          lst.remove(biggest) #This is the faulty line!
09          sort.append(biggest)
10      return sort
```

Explanation 1:

This line is attempts to remove the `biggest` from `lst`. However, variable `biggest` is not defined in the code, the variable that is defined previously is `oldest`.

90. Is the explanation correct? *

Mark only one oval.

- Yes
- No

91. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

92. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
The variable 'biggest' is not defined in the code. It seems there is a typo in line 8, where 'biggest' should be replaced with 'oldest', which is the correct variable representing the oldest person.

93. Is the explanation correct? *

Mark only one oval.

- Yes
- No

94. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

95. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 16

```
01  def sort_age(lst):
02      i=0
03      while i+1<len(lst):
04          if lst[i][1]<lst[i+1][1]:
05              lst.extend([lst[i]])
06              lst.pop(lst[i]) #This is the faulty line!
07          else:
08              i+=1
09      return lst
```

Explanation 1:

This line attempts to remove element from lst using pop. However, this is the incorrect way to use pop() as it can only take in the index of the item and not the tuple of the age. To fix, should use del lst[i] instead.

96. Is the explanation correct? *

Mark only one oval.

- Yes
- No

97. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

98. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
The code tries to remove an element from the list using lst.pop(). However, lst[i] is passed as an argument to lst.pop(), which causes a TypeError. The correct way to remove an element by index is lst.pop(i).

99. Is the explanation correct? *

Mark only one oval.

- Yes
- No

100. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

101. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 17

```
01  def sort_age(lst):
02      agelist = [lst[0],] #This is the faulty line!
03      for i in range(1,len(lst)):
04          if lst[i][1] > agelist[0][1]:
05              agelist.insert(0, lst[i])
06          elif lst[i][1] < agelist[len(agelist)-1][1]:
07              agelist.insert(len(agelist), lst[i])
08          else:
09              for x in range(0,len(agelist)):
10                  if agelist[x][1]> lst[i][1] > agelist[x+1][1]:
11                      agelist.insert(x+1, lst[i])
12                      break
13      return agelist
```

Explanation 1:

This line assumes the input list 'lst' is not empty, leading to an IndexError when 'lst' is empty. A check should be added to ensure 'lst' is not empty before accessing its first element.

102. Is the explanation correct? *

Mark only one oval.

- Yes
- No

103. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

104. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
This line add the first element of `lst` to `agelist`. However, it would generate an error if the `lst` is empty as it try to access element that out of range.

105. Is the explanation correct? *

Mark only one oval.

- Yes
- No

106. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

107. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 18

```
01 def sort_age(lst):
02     for i in range(len(lst)-1):
03         for j in range(len(lst)-i):
04             if lst[j][1] < lst[j+1][1]:
05                 lst[j][1],lst[j+1][1] = lst[j+1][1],lst[j][1] #This is the faulty line!
06             else:
07                 continue
08     return lst
09     pass
```

Explanation 1:

The code tries to swap the ages of two people, but tuples are immutable and cannot be modified. We need to convert the tuples to lists, perform the swap, and then convert them back to tuples.

108. Is the explanation correct? *

Mark only one oval.

- Yes
- No

109. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

110. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
This line is only swapping the ages of the person tuple. However, as the tuple object does not support item assignment, this line would raise a TypeError.

111. Is the explanation correct? *

Mark only one oval.

- Yes
- No

112. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

113. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Evaluation of Explanation for Fault Localization (Part 4)

In this section, you will assess the explanations provided regarding why a specific line of code is deemed faulty. There will be two explanations given for each faulty code. The assessment would focus on the correctness, clearness, and informativeness of the explanation.

The faulty code is extracted from a Python student assignment related to the following task description:
The `top_k` function takes in a list of integers as the input and returns the greatest k number of values as a list, with its elements sorted in descending order.

Faulty Code 19

```
01  def top_k(lst, k):
02      sort = []
03      while lst:
04          largest = lst[0]
05          for i in lst:
06              if i > largest:
07                  largest = i
08          sort.append(i) #This is the faulty line!
09      return sort[:k-1]
```

Explanation 1:
`sort.append(i)` should be `sort.append(largest)` instead. This is because `i` is the loop variable and it will always be the last element in the list, so appending `i` won't give us the largest element.

114. Is the explanation correct? *

Mark only one oval.

- Yes
- No

115. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

116. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
This line is an attempt to append the element `i` from the `lst`. However, `i` would always be the last element from `lst` as it is used for iteration in the previous line. This may be unintended and generate incorrect results since the value of the list would not affect this concatenation.

117. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

118. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

119. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 20

```
1  def top_k(lst, k):
2      a = sort_list(lst) #This is the faulty line!
3      return a[0:k]
```

Explanation 1:
This line calling the `sort_list` function. However, this function is not implemented in the code. This will produce a NameError.

120. Is the explanation correct? *

Mark only one oval.

- Yes
- No

121. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

122. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
The 'sort_list' function is not defined, causing a NameError. To fix this, we can use the 'sorted()' function to sort the list in descending order.

123. Is the explanation correct? *

Mark only one oval.

- Yes
- No

124. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

125. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 21

```
01  def top_k(lst, k):
02      result = []
03      for i in range(len(lst)):
04          largest = lst[0]
05          for element in lst:
06              if element > largest:
07                  largest = element
08          lst.remove(largest)
09          result.append(largest)
10      return lst[:k] #This is the faulty line!
```

Explanation 1:

This line is returning the sublist of `lst`. This is incorrect as `lst` is the inputted list that has not been sorted. Instead it should be returning from `result` which already been sorted.

126. Is the explanation correct? *

Mark only one oval.

- Yes
- No

127. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

128. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
The code incorrectly removes elements from lst during the iteration, resulting in an empty list. This line should be fixed to return the k largest elements from the result list.

129. Is the explanation correct? *

Mark only one oval.

- Yes
- No

130. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

131. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 22

```
01  def top_k(lst, k):
02      lst = sort_descending(lst)
03      return lst[:k-1]
04
05
06  def sort_descending(lst):
07      for i in range(len(lst)-1):
08          for j in range(i, len(lst)):
09              if lst[j][1] > lst[i][1]: #This is the faulty line!
10                  x = lst[i]
11                  lst[i] = lst[j]
12                  lst[j] = x
13      return lst
```

Explanation 1:

The code tries to access the second element of the tuples in the list, but the list contains integers instead of tuples.

132. Is the explanation correct? *

Mark only one oval.

- Yes
- No

133. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

134. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
This line is comparing `lst[j][1]` with `lst[i][1]`. However, `lst` is a list of integer, accessing [1] from `lst[j]` or `lst[i]` will raise an error as it is not subscriptable.

135. Is the explanation correct? *

Mark only one oval.

- Yes
- No

136. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

137. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Evaluation of Explanation for Fault Localization (Part 5)

In this section, you will assess the explanations provided regarding why a specific line of code is deemed faulty. There will be two explanations given for each faulty code. The assessment would focus on the correctness, clearness, and informativeness of the explanation.

The faulty code is extracted from a Python student assignment related to the following task description:
The `search` function receives an input value 'x' and a sorted sequence 'seq' and is responsible for determining the position at which 'x' should be inserted into the sequence while maintaining its sorted order. If 'x' is already present in the sequence, it should return the first possible position for insertion. In the case of an empty 'seq,' it should return 0.

Faulty Code 23

```
01  def search(x, seq):
02      """ Takes in a value x and a sorted sequence seq, and returns the
03      position that x should go to such that the sequence remains sorted """
04      for i in range(len(seq)):
05          if x <= seq[i]:
06              return i
07      return i + 1 #This is the faulty line!
```

Explanation 1:
The line use `i + 1` as the return value. However, this line will cause an error if the seq is empty since 'i' will not have been initialised.

138. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

139. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

140. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Explanation 2:
This line is causing an UnboundLocalError because the variable i is not defined when the loop is not executed. It should be handled separately to return 0 when the sequence is empty.

141. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

142. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

143. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 24

```
01  def search(x, seq):
02      counter = 0
03      for i in range(1, len(seq)+1):
04          if x < seq[i-1]: #This is the faulty line!
05              counter = i-1
06          elif seq[i-1]<x<seq[i]:
07              counter = i
08          counter = i
09      return counter
```

Explanation 1:
This line incorrectly updates the counter when x is less than the current element in the sequence. It should update the counter only when x is greater than or equal to the current element.

144. Is the explanation correct? *

Mark only one oval.

- Yes
- No

145. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

146. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
The condition checks if `x` is less than previous element, if it is, `i-1` would be assigned to `counter`. However, on the line 8 the variable `counter` is assigned with `i` which overwrite the counter that is updated in this line or previous line.

147. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

148. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

149. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 25

```
01  def search(val,seq):
02      if val <= seq[0]: #This is the faulty line!
03          position = 0
04      elif val >= seq[-1]:
05          position = len(seq)
06      else:
07          for item in seq:
08              if val <= item:
09                  position = seq.index(item)-1
10      return position
```

Explanation 1:
This line compares the value 'val' with the first element of the 'seq' sequence. It assumes that 'seq' is not empty, causing an IndexError if 'seq' is empty.

150. Is the explanation correct? *

Mark only one oval.

- Yes
- No

151. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

152. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
This line checks if `val` is smaller than the first element in `seq`. However, if the `seq` is empty, this line would be the cause of the error as it try to access the element from empty `seq`.

153. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

154. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

155. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 26

```
01  def search(x, seq):
02      for i, elem in enumerate(seq): #This is the faulty line!
03          if seq == ():
04              return (i,)
05          elif seq == []:
06              return [i,]
07          elif x <= elem:
08              return i
09          elif x > seq[len(seq)-1]:
10              return len(seq)
```

Explanation 1:
This line is iterating over the element of `seq`. However, if `seq` is empty, there is no return value that is specify outside of the loop. It will return `None` instead of the expected integer type highlighted in line 10.

156. Is the explanation correct? *

Mark only one oval.

- Yes
- No

157. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

158. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
This line is considered suspicious because it iterates over the elements in the `seq` sequence. If the sequence is empty, this loop will not execute, and the subsequent conditions will not be evaluated. This can lead to incorrect results when the sequence is empty.

159. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

160. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

161. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 27

```
01  def search(x, seq):
02      l=len(seq)
03      if x<=seq[0]:
04          return 0
05      elif x>=seq[l-1]:
06          return l+1 #This is the faulty line!
07      else:
08          for i in range (l):
09              if x>=seq[i] and x<=seq[i+1]:
10                  return i+1
11              else:
12                  continue
```

Explanation 1:
The line returns l+1 when x is greater than or equal to the last element in seq. However, the task description specifies that it should return the first possible position if x is found. This line seems to be incorrect.

162. Is the explanation correct? *

Mark only one oval.

- Yes
- No

163. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

164. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
This line checks if `x` is larger than or equal to the last element in `seq` and inserts it at the end of `seq`. However if `x` is equal to the element, it should be inserted in that element index.

165. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

166. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

167. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 28

```
01  def search(x, seq):
02      for i in range(len(seq)): #This is the faulty line!
03          if x > seq[-1]:
04              return len(seq)
05          elif x == seq[i]:
06              return i
07          elif x < seq[i]:
08              return i
```

Explanation 1:
The loop in this line is only proceeded when "seq" is not empty. When "seq" is empty, the function return None while the expected output type is an integer.

168. Is the explanation correct? *

Mark only one oval.

- Yes
- No

169. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

170. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:
There might be an issue with the loop iteration or index calculation.

171. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

172. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

173. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 29

```
01  def search(x, seq):
02      if seq == () or []: #This is the faulty line!
03          return 0
04      else:
05          for i, element in enumerate(seq):
06              for element in seq:
07                  if x > element:
08                      i+=1
09              return i
```

Explanation 1:
`if seq == () or []:` is incorrect. The condition should be `if seq == () or seq == []:` to check if the sequence is empty. The current condition will always evaluate to `False` because `seq == ()` is not a valid syntax.

174. Is the explanation correct? *

Mark only one oval.

- Yes
- No

175. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

176. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:

This line attempts to check the value of seq using the `==` operator along with `()` and `[]`. However, the comparison on an empty list is incorrect since it only uses `[]`, rather than correctly comparing seq with an empty list using `seq == []`.

177. Is the explanation correct? *

Mark only one oval.

- ◯ Yes
- ◯ No

178. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Clear

179. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very	◯	◯	◯	◯	◯	◯	◯	Very Informative

Faulty Code 30

```
01  def search(x, seq):
02      #Takes in a value x and a sorted sequence seq, and returns the
03      #position that x should go to such that the sequence remains sorted
04
05      for i, x in enumerate(seq):
06          if x < seq[i]: #This is the faulty line!
07              return i
08      return len(seq)
```

Explanation 1:
This line checks if `x` is smaller than `seq[i]`. If this condition is satisfied, it will return the index of the item in `seq`. However, value of `x` is reassigned in the 'for' loop that can make this comparison incorrect. Furthermore, this condition check is incomplete. For example, a condition where `x` is found in `seq` is not included.

180. Is the explanation correct? *

Mark only one oval.

- Yes
- No

181. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

182. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Explanation 2:

This line is comparing `x` with `seq[i]`, which is incorrect. The intention is to compare `x` with the element at the current index `i` in the sequence `seq`. However, the variable `x` is being reassigned in the loop header, causing the comparison to be incorrect.

183. Is the explanation correct? *

Mark only one oval.

- Yes
- No

184. How clear is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Clear

185. How informative is the explanation? *

Mark only one oval.

	1	2	3	4	5	6	7	
Very								Very Informative

Open-ended Question