1. Number 1
   1. Parent: Spell 🡪 Child: Accio, Confundo
   2. ***# There seems to be an indentation problem at line 15, 20, and 21. The following is the output when the indentation problem is resolved: #***

Accio

Summoning Charm Accio

No description

Confundus Charm Confundo

Causes the victim to become confused and befuddled

***# If the problem is not resolved, the output will be python raising an error, detecting an indentation error at line 15 ‘Class Confundo(Spell):’ #***

* 1. The get\_description(Confundo()) method that is called is the one in which the class Confundo is defined. This is because the original get\_description() methon in the Spell class is overwritten by the one in the child class Confundo.
  2. ***# After obviously fixing the previous indentation problems raised on point 1b, we need to add the method get\_description() to the Accio class, and return the appropriate description as stated in the question. #***

***# Here is the code after Fixing and adding said method : #***

class Spell:

    def \_\_init\_\_(self, incantation, name):

        self.name = name

        self.incantation = incantation

    def \_\_str\_\_(self):

        return self.name + ' ' + self.incantation + "\n" + self.get\_description()

    def get\_description(self):

        return 'No description'

    def execute(self):

        print(self.incantation)

class Accio(Spell):

    def \_\_init\_\_(self):

        Spell.\_\_init\_\_(self, 'Accio', 'Summoning Charm')

def get\_description(self):

        return 'This charm summons an object to the caster, potentially over a significant distance'

class Confundo(Spell):

    def \_\_init\_\_(self):

        Spell.\_\_init\_\_(self, 'Confundo', 'Confundus Charm')

    def get\_description(self):

        return 'Causes the victim to become confused and befuddled'

def study\_spell(spell):

    print(spell)

spell = Accio()

spell.execute()

study\_spell(spell)

study\_spell(Confundo())

***# Changes made is highlighted in purple. #***

1. Number 2
2. from os import system, name
3. from statistics import mean
4. class Employee:
5. # Init variables
6. \_\_id: str
7. \_\_name: str
8. \_\_pos: str
9. \_\_salary: int
10. employees = []
11. def \_\_init\_\_(self, id: str, name: str, pos: str, salary: int):
12. self.\_\_id = id
13. self.\_\_name = name
14. self.\_\_pos = pos
15. self.\_\_salary = salary
16. # Getter
17. def getId(self) -> str: return self.\_\_id
18. def getName(self) -> str: return self.\_\_name
19. def getPos(self) -> str: return self.\_\_pos
20. def getSalary(self) -> int: return self.\_\_salary
21. # Setter
22. def setId(self, id: str): self.\_\_id = id
23. def setName(self, name: str): self.\_\_name = name
24. def setPos(self, pos: str): self.\_\_pos = pos
25. def setSalary(self, salary: int): self.\_\_salary = salary
26. # Funcs
28. def \_\_str\_\_(self):
29. return "ID: {}, Name: {}, Position: {}, Salary: {}".format(self.getId(), self.getName(), self.getPos(), self.getSalary())
30. def clear():
31. if name == 'nt':
32. \_ = system('cls')
33. else:
34. \_ = system('clear')
35. def fillRightWith(amount: int, string: str, filler: str) -> str:
36. return ( string + filler \* (amount - len(string)) ) if amount > len(string) else string
37. def showEmployees():
38. print("|ID    |Name                 |Position       |Salary               |")
39. for i in Employee.employees:
40. dispId = fillRightWith(6, i.getId(), " ")
41. dispName = fillRightWith(21, i.getName(), " ")
42. dispPos = fillRightWith(15, i.getPos(), " ")
43. dispSalary = fillRightWith(21, str(i.getSalary()), " ")
44. print( "|{id}|{name}|{pos}|{salary}|".format(id=dispId, name=dispName, pos=dispPos, salary=dispSalary) )
45. def prompt() -> str:
46. print("1. New Staff")
47. print("2. Delete Staff")
48. print("3. View Summary Data")
49. print("4. Save & Exit")
50. print()
51. return input("Input choice(numeric): ")
53. with open("data.txt", "r", encoding="utf-8") as file:
54. rawData = file.readlines()
55. for i in range(len(rawData)):
56. rawData[i] = rawData[i].replace("\n", "").split("#")
57. person = rawData[i]
58. Employee.employees.append(Employee(person[0], person[1], person[2], person[3]))
59. while True:
60. clear()
61. showEmployees()
62. print()
63. answer = prompt()
64. if answer == "1":
65. print()
66. print("New Staff:")
67. while True:
68. newId = input("Input ID[SXXXX]: ")
69. if newId[0] != "S":
70. continue
71. for i in Employee.employees:
72. if i.getId == newId:
73. continue
74. try:
75. int(newId[1:])
76. except:
77. continue
78. newName = input("Input Name[0..20]: ")
79. if not (len(newName) > 0 and len(newName) < 21):
80. continue
82. allPos = ["Staff", "Officer", "Manager"]
83. newPos = input("Input Position[Staff|Officer|Manager]: ").capitalize()
84. if newPos not in allPos:
85. continue
86. newSalary = int(input("Input salary for " + newPos + ": "))
87. if newPos == "Staff":
88. if newSalary < 3500000 or newSalary > 7000000:
89. continue
90. elif newPos == "Officer":
91. if newSalary < 7000001 or newSalary > 10000000:
92. continue
93. else:
94. if newSalary < 10000001:
95. continue
97. Employee.employees.append(Employee(newId, newName, newPos, newSalary))
98. print("Add employee success !")
99. input("Press [ENTER] to continue..")
100. elif answer == "2":
101. print()
102. delete = input("Input ID: ")
103. for i in range(len(Employee.employees)):
104. if Employee.employees[i].getId == delete:
105. Employee.employees.pop(i)
106. print("Delete successful !")
107. input("Press [ENTER] to continue..")
108. continue
109. print("Invalid input !")
110. input("Press [ENTER] to continue..")
111. elif answer == "3":
112. staffSal = [100000000, 0]
113. staffMean = []
114. officerSal = [100000000, 0]
115. officerMean = []
116. managerSal = [10000000, 0]
117. managerMean = []
118. for i in Employee.employees:
119. if i.getPos == "Staff":
120. if i.getSalary() < staffSal[0]:
121. staffSal[0] = i.getSalary()
122. if i.getSalary() > staffSal[1]:
123. staffSal[1] = i.getSalary()
124. staffMean.append(i.getSalary())
125. elif i.getPos == "Officer":
126. if i.getSalary() < officerSal[0]:
127. officerSal[0] = i.getSalary()
128. if i.getSalary > officerSal[1]:
129. officerSal[1] = i.getSalary()
130. officerMean.append(i.getSalary())
131. else:
132. if i.getSalary() < managerSal[0]:
133. managerSal[0] = i.getSalary()
134. if i.getSalary() > managerSal[1]:
135. managerSal[1] = i.getSalary()
136. managerMean.append(i.getSalary())
137. clear()
138. print("1. Staff")
139. prompts = ["Minimum Salary: ", "Maximum Salary: ", "Average Salary: "]
140. for i in range(len(prompts)):
141. print(prompts[i], end='')
142. if i == 0:
143. print(staffSal[0])
144. elif i == 1:
145. print(staffSal[1])
146. else:
147. print(mean(staffMean))
148. print("2. Officer")
149. for i in range(len(prompts)):
150. print(prompts[i], end='')
151. if i == 0:
152. print(officerSal[0])
153. elif i == 1:
154. print(officerSal[1])
155. else:
156. print(mean(officerMean))
157. print("3. Manager")
158. for i in range(len(prompts)):
159. print(prompts[i], end='')
160. if i == 0:
161. print(managerSal[0])
162. elif i == 1:
163. print(managerSal[1])
164. else:
165. print(mean(managerMean))
167. input("Press [ENTER] to continue..")
169. elif answer == "4":
170. for i in Employee.employees:
171. with open("data.txt", "w", encoding="utf-8") as file:
172. file.write("{}#{}#{}#{}\n".format(i.getID(), i.getName(), i.getPos(), str(i.getSalary())))
173. print("Save successful !")
174. input("Press [ENTER] to continue..")
175. else:
176. print("Invalid input.")
177. input("Press [ENTER] to continue..")