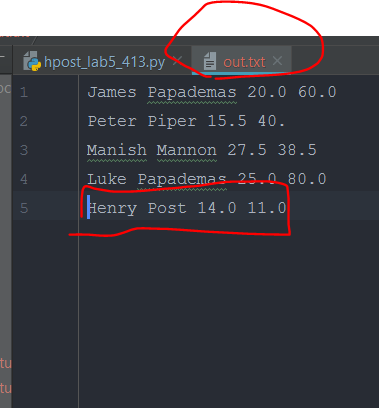
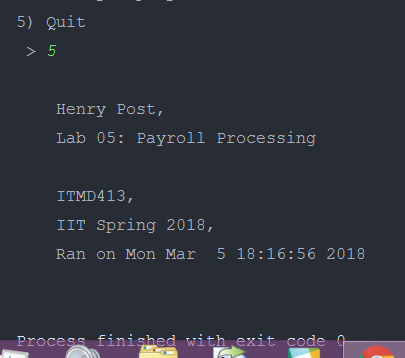
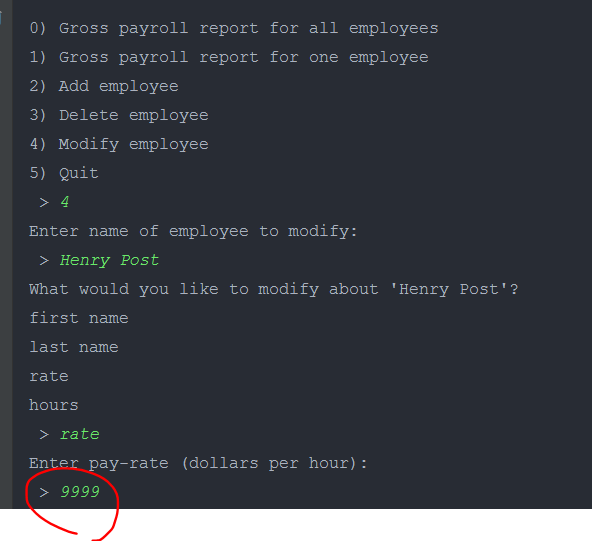
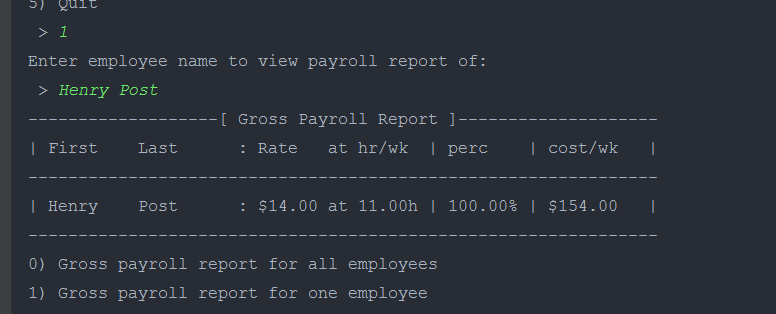
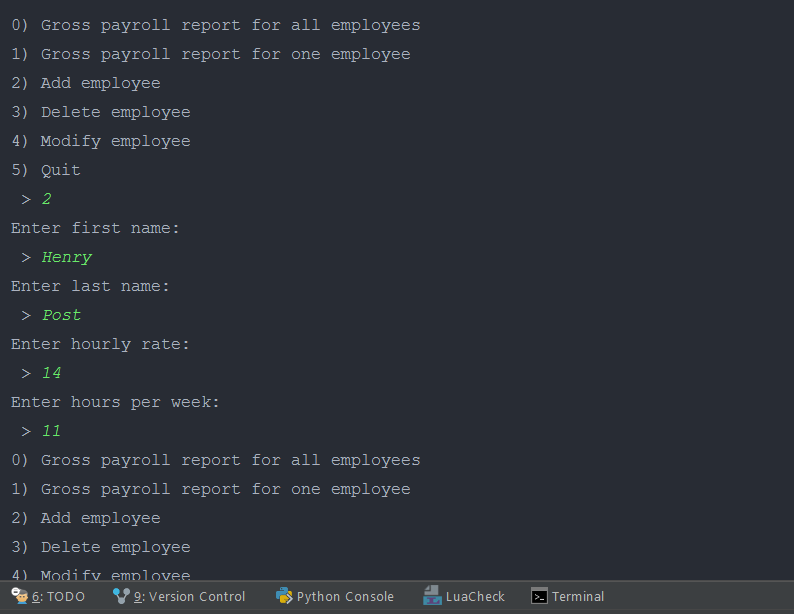
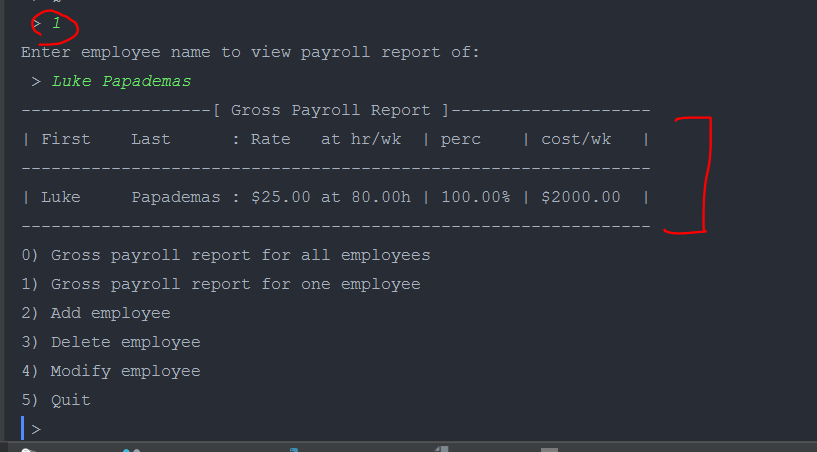
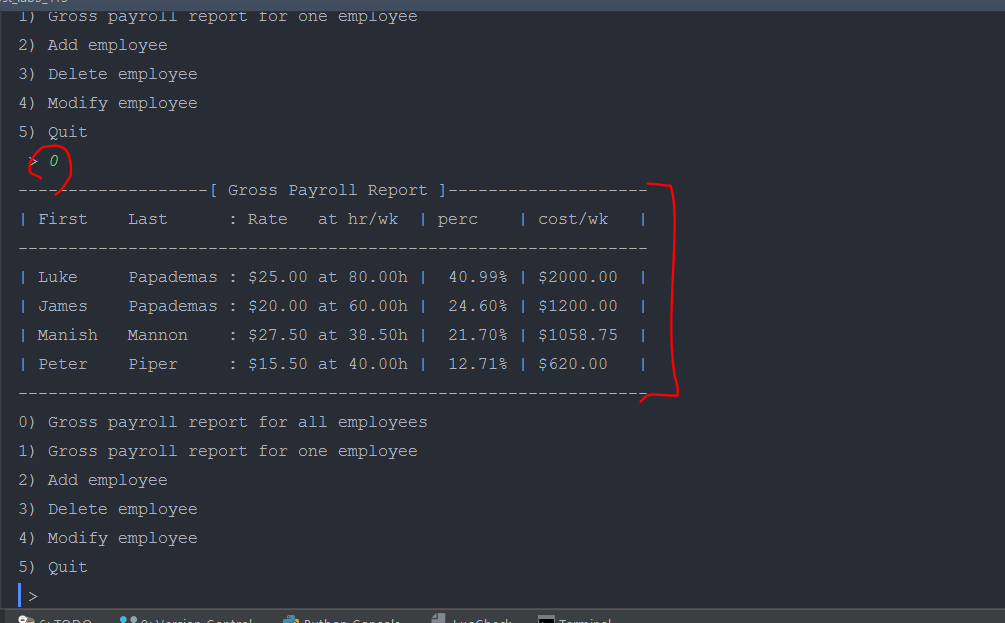
Henry Post

ITMD413

Lab 5, Payroll Processing



1. **import** time
2. **from** copy **import** deepcopy
4. \_\_info\_\_ = \
5. f"""
6. Henry Post,
7. Lab 05: Payroll Processing
9. ITMD413,
10. IIT Spring 2018,
11. Ran on {time.strftime("%c")}
12. """
14. \_option\_gross\_all = 0
15. \_option\_gross\_one = 1
16. \_option\_add = 2
17. \_option\_delete = 3
18. \_option\_modify = 4
19. \_option\_quit = 5
21. \_options = {}
23. \_options[\_option\_gross\_all] = "Gross payroll report for all employees"
24. \_options[\_option\_gross\_one] = "Gross payroll report for one employee"
25. \_options[\_option\_add] = "Add employee"
26. \_options[\_option\_delete] = "Delete employee"
27. \_options[\_option\_modify] = "Modify employee"
28. \_options[\_option\_quit] = "Quit"

31. **class** Employee:
32. **def** \_\_init\_\_(self, fname, lname, rate, hours):
33. self.fname = fname
34. self.lname = lname
35. self.rate = rate
36. self.hours = hours
38. **def** to\_string(self):
39. **return** " ".join([self.fname, self.lname, str(self.rate), str(self.hours)])
41. **def** name(self):
42. **return** self.fname + " " + self.lname
44. **def** \_\_str\_\_(self):
45. **return** f"{self.fname:8s} {self.lname:10s}: ${self.rate:5.02f} at {self.hours:5.02f}h"
47. **def** weeklyRate(self):
48. **return** self.rate \* self.hours
50. **def** modify\_first\_name(self):
51. **while** True:
52. **try**:
53. fn = input("Enter first name:\n > ")
54. self.fname = fn
55. **return**
56. **except** Exception as e:
57. **print**("Invalid first name.")
59. **def** modify\_last\_name(self):
60. **while** True:
61. **try**:
62. ln = input("Enter last name:\n > ")
63. self.lname = ln
64. **return**
65. **except** Exception as e:
66. **print**("Invalid last name.")
68. **def** modify\_rate(self):
69. **while** True:
70. **try**:
71. r = input("Enter pay-rate (dollars per hour):\n > ")
72. self.rate = float(r)
73. **return**
74. **except** Exception as e:
75. **print**("Invalid rate.")
77. **def** modify\_hours(self):
78. **while** True:
79. **try**:
80. h = input("Enter hours worked per week:\n > ")
81. self.hours = float(h)
82. **return**
83. **except** Exception as e:
84. **print**("Invalid hours.")
86. **def** modify(self):
87. """Prompt the user to modify this employee object."""
88. **print**(f"What would you like to modify about '{self.name()}'?")
90. choices = {
91. "first name".upper(): self.modify\_first\_name,
92. "last name".upper(): self.modify\_first\_name,
93. "rate".upper(): self.modify\_rate,
94. "hours".upper(): self.modify\_hours,
95. }
97. **for** item, func **in** choices.items():
98. **print**(item.lower())
100. answer = input(" > ").upper()
102. **while** True:
103. **try**:
104. **return** choices[answer]()
105. **except** Exception as e:
106. **print**("Invalid choice.")
107. answer = input(" > ")

110. **def** highest\_cost(employees):
111. """
112. Given a list of Employee objects, return the index of the one that costs the most.
113. """
114. x = 0
116. highest\_e: Employee = employees[x]
118. **for** i **in** range(len(employees)):
119. **if** employees[i].weeklyRate() > employees[x].weeklyRate():
120. x = i
122. **return** x

125. **def** sort\_by\_highest\_cost(employees):
126. """
127. Given a list of Employee objects, sort them by their costs, high to low.
128. """
129. sorted = []
130. emps = deepcopy(employees)
132. **while** (emps):  # while original list has elements
133. x = highest\_cost(emps)  # get pos of highest
134. e = emps[x]  # record it
135. **del** emps[x]  # delete it
136. sorted.append(e)  # append it
138. **return** sorted

141. **def** print\_names(employees):
142. names = [(e.fname + " " + e.lname) **for** e **in** employees]
143. names.sort()
145. **for** name **in** names:
146. **print**(name)

149. **def** employee\_by\_name(name: str, employees):
150. **while** "  " **in** name:  # flatten down ALL multiple spaces
151. name = name.replace("  ", " ")
153. **try**:
154. narr = name.split(" ")[0:2]
155. fname, lname = narr
156. **except** ValueError as e:
157. **return** None
159. **for** e **in** employees:
160. e: Employee
161. **if** e.fname == fname **and** e.lname == lname:
162. **return** e

165. **def** delete\_employee\_object(employees, employee):
166. """
167. Given a list of Employees and an Employee, delete the Employee.
168. """
169. **for** i **in** range(len(employees)):
170. **if** employees[i] **is** employee:
171. **del** employees[i]
172. **return** employee

175. **def** payroll\_report(employees):
176. """
177. Given a list of Employee objects, print an overview of payroll data.
178. """
180. employees = sort\_by\_highest\_cost(employees)
182. er\_strs = {}  # employee report strings
183. totalcost = 0.0
185. **for** e **in** employees:  # get total cost of paying ALL employees
186. e: Employee
188. totalcost += e.weeklyRate()  # for averages or percents later
190. header = ("| {:8} {:9} : {:6} at {:6} | {:7} | {:9} |".format("First", "Last", "Rate", "hr/wk", "perc", "cost/wk"))
191. div = ''.join("-" **for** i **in** range(len(header)))
193. **print**(sprint\_center(div, "[ Gross Payroll Report ]"))
194. **print**(header)
195. **print**(div)
197. **for** e **in** employees:
198. e: Employee
200. percentCost = e.weeklyRate() / totalcost  # how much does person X cost out of 100%?
202. s = "| "  # a report string unique to that employee
204. s += str(e) + " | "  # add the employee's normal data
205. s += f"{percentCost\*100:6.02f}% | "  # add their percentage cost
206. s += f"${e.weeklyRate():<8.02f} | "  # add how much the employee costs per week
208. **print**(s)
210. **print**(div)
212. **return** employees

215. **def** parse\_employee\_line(line):
216. """
217. Turn a single line of text into an Employee.
218. """
219. fname, lname, rate, hours = line.split(" ")[0:4]
221. **return** Employee(fname, lname, float(rate), float(hours))

224. **def** employees\_to\_files(employees, path):
225. """
226. Given a list of Employees, write them to a file at ``path``.
227. """
228. file = open(path, "w+")
230. e: Employee
231. lines = [e.to\_string() **for** e **in** employees]
233. **for** line **in** lines:
234. file.write(line + "\n")
236. file.close()

239. **def** sprint\_center(src, innie):
240. """
241. Given a src string, put 'innie' in the middle of it.
242. Example: sprint\_center("+---\_---+","hey") -> "+--hey--+"
243. """
245. # print(f"Wanna put '{innie}' inside of '{src}'")
247. src = [char **for** char **in** src]
248. innie = [char **for** char **in** innie]
250. mid = len(src) // 2
252. in\_pos = mid - (len(innie) // 2)
254. j = 0
255. **for** i **in** range(in\_pos, in\_pos + len(innie)):
256. src[i] = innie[j]
257. j += 1
259. **return** ''.join(src)

262. **def** parse\_employee\_txt(filepath):
263. """
264. Parse a flat text file representing a list of Employees.
265. :param filepath: The location of the file.
266. :return: A list of Employees.
267. """
268. emps = []
270. with open(filepath, "r") as file:
271. **for** line **in** file:
272. emps.append(parse\_employee\_line(line))
274. **return** emps

277. **def** payroll\_report\_all\_employees(employees):
278. payroll\_report(employees)

281. **def** payroll\_report\_one\_employee(employees):
282. n = input("Enter employee name to view payroll report of:\n > ")
284. e = employee\_by\_name(n, employees)
286. **if** e **is** **not** None:
287. payroll\_report\_all\_employees([e])
288. **else**:
289. **print**(f"Name '{n}' not found.")
291. print\_names(employees)

294. **def** add\_employee(employees):
295. **while** True:
296. **try**:
297. fname = input("Enter first name:\n > ")
298. **break**
299. **except** Exception as e:
300. **print**("Invalid first name.")
302. **while** True:
303. **try**:
304. lname = input("Enter last name:\n > ")
305. **break**
306. **except** Exception as e:
307. **print**("Invalid last name.")
309. **while** True:
310. **try**:
311. rate = float(input("Enter hourly rate:\n > "))
312. **break**
313. **except** Exception as e:
314. **print**("Invalid hourly rate.")
316. **while** True:
317. **try**:
318. hours = float(input("Enter hours per week:\n > "))
319. **break**
320. **except** Exception as e:
321. **print**("Invalid hours per week.")
323. employees.append(Employee(fname, lname, rate, hours))

326. **def** delete\_employee(employees):
327. n = input("Enter name of employee to delete:\n > ")
329. e = employee\_by\_name(n, employees)
331. **if** e:  # if we found an employee by that name
332. delete\_employee\_object(employees, e)  # delete it
333. **print**(f"Employee '{n}' deleted.")
334. **else**:  # employee not found.
335. **print**(f"Employee by name '{n}' not found.")

338. **def** modify\_employee(employees):
339. n = input("Enter name of employee to modify:\n > ")
341. e = employee\_by\_name(n, employees)
343. **if** e:
344. e.modify()  # modify self
345. **else**:
346. **print**(f"Employee by name '{n}' not found.")

349. \_optionfns = {}
351. \_optionfns[\_option\_gross\_all] = payroll\_report\_all\_employees
352. \_optionfns[\_option\_gross\_one] = payroll\_report\_one\_employee
353. \_optionfns[\_option\_add] = add\_employee
354. \_optionfns[\_option\_delete] = delete\_employee
355. \_optionfns[\_option\_modify] = modify\_employee
356. \_optionfns[\_option\_quit] = **lambda** x: (
357. **print**(\_\_info\_\_),
358. employees\_to\_files(x, "out.txt"),
359. exit(0))
361. **if** \_\_name\_\_ == '\_\_main\_\_':
363. **print**("Hello!")
365. employees = parse\_employee\_txt("./employees.txt")
367. **for** employee **in** employees:
368. **print**(employee)
370. choice = -1
372. **while** True:
373. **if** choice **is** \_option\_quit:
374. **print**("Goodbye!")
375. **break**
377. **for** i **in** \_options:  # display the options
378. option = \_options[i]
379. **print**(f"{i}) {option}")
381. choice = int(input(" > "))  # get choice
382. **if** choice **in** \_optionfns:
383. \_optionfns[choice](employees)  # execute choice
384. **else**:
385. **print**(f"'{choice}' isn't a valid option.")
387. **print**(\_\_info\_\_)