目次へ

指南編(残業時間2)

最初のかた

できた。

```
gd={}
for s in open("leavetime2.txt"):
  list=s[:-1].split(" ")
  name=list[0]
  time=list[1].split(":")
  hour=int(time[0])
  minute=int(time[1])
  zangyo=(hour*60+minute)-(17*60+20)
  if zangyo<0:
    zangyo=zangyo+(24*60)
  gd.setdefault(name, 0)
  gd[name]=gd[name]+zangyo
for i in gd:
   print i, int(gd[i])/60, "hours", int(gd[i])%60, "minutes"
```

実行してみましたが、正常に動きますね。たいへんお疲れ様でした。

最初から見ていきます。

gdという辞書を使って統計処理をしてくれました。筆者が例に挙げた変数名ですね。参考までに、あの例では、「goukei の dictionary」という程度の安易な命名でした。

で、listという変数に、おしまい一文字（改行文字）を取り除いたあとで空白記号で分割したリストを入れました。実は、「list」という変数名はpythonにとって別の機能をすでに割り当てられているため、それを塗りつぶしてしまいます。これで動いちゃうところがpythonの面白いところでもありますが、この命名は避けておくのが賢明と思います。筆者なら、listの代わりに、a_listとか、alistとでもするかな。

timeという変数名も、実はちょっと怖いです。timeは標準モジュールとして存在しますので、このモジュールをあとで使う予定のときは避けるとよいでしょう。モジュールが何かという説明は、きっといつか。

もっとも、こういう変数名の「かぶり」については、後に不都合があきらかになった時点で一括で変えてしまえばよいだけですし、経験の問題です。

スクリプトを調べるところに戻ります。nameには分割後の左半分を、hourとminuteには分割後の右半分をさらに「:」で分割して、それの一つずつを入れています。

zangyoには17時20分を超えた時間（分）を算出、ゼロを下回るときは翌日補正ということで24時間分の値を追加。

setdefaultの使いどころも適切です。ここでgdがnameをキーとして必ず持つようにさせるから、直後のgd[name]という指定でエラーを起こす可能性がなくなるわけです。

統計を算出する処理はまったくOKのようです。

で、結果の表示部分ですね。gd（辞書）についてforを使うと、キーにあたる値が各々取得できます。（っていうか、できるんですね。筆者は実は初めて知りました。keys をつかってあらかじめキー一覧をリストとして取得しておくのがクセになっていましたので。へえ、そうなのかあ。）

で、辞書の中には分換算の残業時間がたまっていますから、こいつを「何時間、何分」というフォーマットに直して表示させてくれました。結果が見やすいです。

これで全成員の残業時間統計が一行ずつすべて表示されますから、あとは誰がトップかを目で判断すればよいようになっています。問題文でもそれで充分としていたのだから、これでよいです。

実は最終行は、print i, gd[i]/60, "hours", gd[i]%60, "minutes"　だけにしてもいいですよ。int( )でそれぞれの値を囲んでくれましたが、gd辞書の中身（キーに対応するほう）は、数字として管理されていますので。文字列なら明示的にint( )するところですが、ここは必ず数字が入っていると前提してよいのです。

ともあれ、実行結果はばっちりです。お見事でした。

べつのかた

できた。

```
ruikeibo = {}
SHUGYO = 17*60 + 20

for my_line in open ("leavetime2.txt"):
    my_pair = my_line[:-1].split(" ")
    taishaJikoku = my_pair[1].split(":")
    zangyo  = int(taishaJikoku[0])*60 + int(taishaJikoku[1]) - SHUGYO
    if zangyo < 0:
        zangyo = zangyo + 24*60

    ruikeibo.setdefault(my_pair[0], 0)
    ruikeibo[ my_pair[0] ] += zangyo

saicho = 0
for namae in ruikeibo:
    if saicho < ruikeibo[namae]:
        longest_person = [namae]
        saicho = ruikeibo[namae]
    elif saicho == ruikeibo[namae]:
        longest_person.append(namae)

print longest_person, ": ", saicho/60, "hours", saicho%60, "minutes"
```

これを実行すると、最大時間の人だけが出力されます。先取りですねえ、いいですね。

残業時間を累積させていくのに使う辞書を、ruikeiboと分かりやすい名前に設定しました。打ち込むのは面倒になったでしょうが、読むにはとてもよいです。

その次に、SHUGYOという変数に、あらかじめ「17時20分」の経過時間（分）を計算しておいて、それを以後使おうということにしてくれました。変数は小文字にしておくとよいよ、と以前書きましたが、ここではお決まりの値に別名をつけておくという「定数」の使い方に近いので、大文字にするという慣習（たしかにあるんです、こういう慣習）に倣うというのはアリですね。こやつできる。　また、こういう書き方にしておくと、あとで「残業時間はやっぱ17時40分ね」とかいう事態になったときの対処もしやすいのです。

さて、ruikeibo変数に統計結果ができあがったところで、それのレポートを出力する部分に入ります。コード後半。saicho変数に「暫定トップ残業時間」、longest_personに「暫定トップ残業者＋月」を入れることにして、ruikeiboに溜まったキーと値を順に評価しながら最長記録を探していきました。

longset_personに、ただの文字列を入れないというところが、筆者の意図を超えて鋭いです。「最長時間タイ記録」があるかも知れないと踏んだわけですね。もしも「Yamamotoさん、6月」の記録と「Watanabeさん、12月」が同じときは、記録を完全に塗り替えたり、あるいは無視したりする代わりに、同一記録はリストに順々に併記されていくという表現になりました。

いやあ、できすぎですよ。

あ、コードの途中にある「+=」という記号の使い方は、テキストの中ではまだ説明してなかったかな。こいつは何だろうと思った人へ。 a = a + 1 といった形に相当するときは、a += 1 という書き方が許されています。こうすると、「ああ、1増やすんだね」ということがより明確に表現

できる、こともあります。同様に、「-=」「*=」「/=」なんかも使えますので、興味がある人は対話シェルとかでいろいろ実験してみましょう。

つぎのかた

出遅れた人も、なんか送ってくれればここに紹介してまいります。ほぼ完成形が出てしまいましたが。