目次へ

練習問題・残業時間

練習問題編では、「仕事にひょっとして役にたつっぽい、それらしい使い道」というのを意識しながら進めていきます。

まずは、退社時間を見て、定時からどれだけの残業量かを計算してみる、という仕事を考えてみることにしましょう。あまり愉快な話ではないですが、毎日のタイムカードの値から残業代を計算したりするのに使えそうなネタじゃないでしょうか。まあ、単純な計算だけで済まない色々な追加ルールがあることは承知していますが、あくまで最初のとっかかりとして。

準備

さて、最初は、いきなりスクリプトを書き始めずに、いろいろと頭の中で考えを巡らせてみることが重要です。今から何をしようとするのか、ということをハッキリさせるためです。

たとえば、ある会社において、定時が午後5時20分だったとする。Aさんの退社時間は午後7時15分だった。残業時間は何時間何分か。

まずは、頭でどうやって考えているかを注意してたどってみましょう。『「時」だけで考えてみると、5時から7時ってことは2時間くらいの残業か。これに「分」のことを加味してみると、20分から15分だから、実際には2時間には(15-20 = -5)ってことで、5分ほど足りない。じゃあ1時間55分ってところかな…』

ですが、今みたいな考え方を直接プログラムに直してみようとすると、どう書いたものかな、とちょっと悩んでしまいそうです。もうちょっとプログラムに落としやすい考え方に直してみましょ

う。コツは、単純な計算に直せそうかを探すことです。

たとえば、午後5時20分を、午前とか午後とかナシで、17時20分と呼んでみる。こうすると、午前でも午後でも統一的に計算できそうです。また、17時20分を、「0時0分から何分経った瞬間か」ということに置き換えてみる。(17*60 + 20) ということで、1040。17時20分は、「零時から1040分目」です。これに対して、退社時間の午後7時15分は、(19*60 + 15)で、「零時から1155分目」です。

（今、筆者は、これを書きながら、（17*60 + 20）とか(19*60 + 15)とかの計算をpythonの対話シェルを使って計算しています。どうでもいいけど）

ここまでいじると、残業時間は、単なる引き算に直すことができます。すなわち 1155 - 1040 ですね。答えは115。60で割ると、1余り55。ということで、残業時間は1時間55分でした。

うん、このくらい明確なルールに直せたら、プログラムとしても表現できそうな気がしてきます。この線でいきましょう。（残業が深夜をまわって0時何分とかになったらどうするの、というケースは、今回は考慮していません。興味があったら考えてみましょう）

115分が1時間15分であることを計算する方法は、今のうちに確認しておきましょう。pythonで、割り算の商と余りを出す方法は思い出せるでしょうか。対話シェルなら、下のような感じですよ。

```
>>> 115 / 60
1
>>> 115 % 60
55
```

この、パーセント記号を使う不思議な計算を思い出してくださいね。「/」では（小数を切り捨てた）商、「%」では割り算の余りが出るのでした。

115という数字は、「17時20分」を「1040」に、「19時15分」を「1155」に直して引き算した結果でした。じゃあ今度は、「17時20分」を「1040」に直す方法も確認します。たとえばhという変数に「時」、mという変数に「分」が入っているとしたら、下のような表現でいけそうですね。

```
>>> h = 17
>>> m = 20
>>> h*60 + m
1040
```

簡単です。

ここは簡単ですが、次はちょっと頭をひねります。"17:20"という「文字列」が与えられたとしたら、これを17時20分なのだと判断するためにどうしたらよいでしょうか。

なんでそんな面倒なことを考え出さなくちゃいけないんだ、と思うでしょうか。でも、これができるようにしておくと、あとで人間が使いやすいプログラムにすることができるのですよ。完成した形を見ればわかる…と思います。

ファイルの扱い

でも一旦この話題を中断して、新しいことに移ります。スクリプト以外の外部ファイルを読み込む仕事のやりかたです。今までにまだ触れたことのない話題です。

たいていの役に立つプログラムは、外からなにか材料を持ってきて、それを何らかの形で利用するものです。「外」といってもいろいろあります。人間がなにか入力するのも「外」だし、カメラから画像を読み込んで何かするのも「外」だし。インターネットのどこかから決まったページを読み込んでくるのも「外」です。

今回の「外」は、一番簡単なパターンで、ファイルです。ファイルの中でも、一番扱いやすい、テキストファイルです。テキストファイルというのが正確に何なのかはまだきっちり説明できませんが、直観的には「メモ帳」で開いて中身が普通に確認できるファイルです。

ファイルを使うときは、対話シェルを使うと（「現在位置」の都合上）ちょっとやりにくいところがありますので、スクリプトファイルをちゃんと作っていろいろ試すことにしましょう。スクリプトファイルもテキストファイル、別に読み込もうとするネタもテキストファイル。二つのファイルがそろってはじめて試せることです。

まずは、下のようなテキストファイルを書いて、今からスクリプトをつくろうとするフォルダの中に保存しておきましょう。ファイル名は data.txt とでもしましょう。下のファイル名のあたりをクリックしてダウンロードしてもよいです。

data.txt

```
first line
second line
last line
```

このファイルと同じ場所に、下のようなpythonのスクリプトファイルをつくります。ファイル名は、dataread.py とでもします。スクリプトファイルをつくって編集する方法は以前説明したので、なんとかがんばってやってみてください。

```
f = open("data.txt")
s = f.read()
print s
```

これをコマンドプロンプトから実行します。コマンドプロンプトを開いて、「現在位置」をさっきのスクリプトの場所に移動して、スクリプトファイルを実行する、という手順も、前に説明したとおりです。なんとかがんばってやってみてください。こればっかりですが。

うまくいくと、実行結果として下のような表示が確認できるでしょう。

```
c:¥work>dataread.py     ←「現在位置」は、環境によって違います
first line
second line
last line


c:¥work>
```

外部ファイルを使うはじめての例でした。うまくいきましたでしょうか。

今作ったスクリプトの中身を解説します。特に一行目と二行目。

```
f = open(″data.txt″)   ← f に何を入れた?
s = f.read()      ← s に何が入る?
print s
```

一行目。f という変数に何かを入れています。変数にはいろいろなものが入ることは見てきました。数字も入るし、文字列も入る。リストとか、辞書とか、そんなものも入りました。ここでは、さらに別の種類の何かを入れています。「ファイル」です。もうちょっと正しくは、「ファイルを開いていますよ、という目印」です。

open("ファイル名") という書き方で、pythonはそのファイルを開いて、今から読み込めますよ（まだ読み始めてないけど）という、「準備オッケー」な状態をつくってくれるわけです。で、使う人はその「オッケーな状態」に対して具体的な処理を実行することができる、と。

うまく説明できているか自信がありません。きっとここは、非プログラマだった人にとっては結構な飛躍が要求される場所だと思うのです。

変な例えになるかもしれませんが、牛乳の入ったテトラパックを想像しましょう。これがひとつのファイルです。openという命令を受けて、pythonはストローをそのパックにぶっ刺す。まだ牛乳はストローから出てきません。この状態で、ストローから中身を吸い出してみるという命令をうけるまではこのままです。変数に入るのは、このストローの先っちょ部分の目印、という…　だめかなあ、余計混乱するかなあ。

まあ、まずはこんなところとしておいて、二行目の説明をします。

この書き方で、sという変数には、さっきの「ストローの口」から中身をずるっと吸い出した中身が入ります。変数に、「.read()」という命令をくっつけることで、ここから読むんだということをハッキリさせています。sの中身は、文字列です。ファイルから読み出したものは、文字列になります。テキストファイルの中には確かに文字が書いてあったし、ここは自然でしょ。

でも、print s として一回だけprintしてみたのに、実行結果ではいきなり3行分の出力があったのに驚いた人もいるかもしれません。スクリプトの三行目のこと。s の中には、別に三つの文字列がいっぺんに入ったわけではありません。改行も含めて全部を、ひとつの文字列として扱ったというまでのことです。

実は、「改行」というのも、コンピュータ的にはひとつの文字なんです。「改行記号」とか「改行文字」とか呼ぶこともあります。ためしに、対話シェルで下のようなことを試してみましょう。

```
>>> print ″a¥nb″
a
b
```

"\n"（\が使えない人は¥で代用）は、改行記号をプログラム中で表現するためのワザです。実際にprintしてみると、なにかを表示する代わりに改行をしていることがわかりますね。

.read() という書き方は、改行も何もすべて含めて、ファイルの中身を全部ひとつの文字列として読み込んでしまう書き方なんです。めちゃくちゃ大きいファイルに対して .read() をすると、ものすごいサイズの文字列が発生してパソコンの処理能力をひどく浪費させることになりますのでご注意。

.read() は今後あまり使うことはないでしょう。かわりに、扱う対象がテキストファイルなら、「一行ずつ読み込む」という書き方も準備されていますから、こっちを使うのがより便利で、たいていの場合は正しい対処です。

さっきつくった readdata.py を、下のように書き換えてください。

```
for s in open("data.txt"):
  print s
```

これを実行すると、間違いがなければ、下のような表示がされます。

```
c:¥work>dataread.py
first line

second line

last line


c:¥work>
```

実行結果はさっきと似たような感じです。ですが、この書き方は、一行だけ読み込んで表示、次の一行を読み込んでまた表示、という繰り返しを行います。さっきの牛乳パックの例えをまた持ち出すなら、一気に全部吸ってしまうのが .read() の方法、必要な量だけを少しずつ吸って仕事をするのが今回の方法です。こっちの方法なら、巨大なファイルがあったとしても、時間はかかっても、一度にやるべき仕事はわずかですから、コンピュータに優しいです。

このfor...とかいう書き方は、今のところはお決まりの書き方として丸呑みに覚えてしまっていてもいいでしょう。「なんか、リストの中身を順番に見るときに、似たような書き方があったなあ」と思い出してくれるなら上等です。（イテレータとか何とか、そんな難しい用語はずっと後。）

表示が縦にスカスカに見えている理由を簡単に説明しておきます。「テキストファイルを一行ずつ読み込んでいる」といいました。一行の終わりには、改行文字が含まれます。print命令ってのは、今までは意識しないですみましたが、何かを表示したあと、改行文字もひとつ付け足しで表示するという動作をこっそりやってくれていたのです。「一行分」の終わりにいつも含まれる改行文字と、print命令が暗黙に付け足してくれる改行文字がかぶってしまったので、プログラムの実行結果は、改行がいちいち余分にされる感じになってしまったわけです。

これを防ぐには、さっきの readdata.py を下のように直しましょう。

```
for s in open("data.txt"):
  print s[:-1]
```

おわかりでしょうか。「文字列の最後の一文字だけを除く」という書き方を覚えていましたか。最後の一文字がいつも改行文字だというなら、こいつを削ってしまえばよいわけですね。実行結果は、確かめておいてください。

なんか、残業時間を云々とかいいながら、気がつけばテキストファイルの読み方のためにずいぶん長い寄り道をしています。

一応理由はあって、次のような練習問題を考えているからです。すなわち、**テキストファイルにたくさんの行が含まれていて、これは誰かの毎日の退社時刻である。総残業時間を求めよ** って感じ。

一行ずつ、残業時間を調べる

ということで、やっと本題に戻ります。

今から、準備されたあるファイルを読んで、残業時間の合計を出そうというのがここでの主題なのです。こんなファイルが用意されたと想像してください。

```
17:30
18:35
17:20
19:05
22:30
17:20
つづく…
```

テキストファイルを一行ずつ読み込む方法はさっきやりましたので、次は "17:30" という文字列をちゃんとpythonが扱える形に加工することを考えます。

17という数字と、30という数字があるのなら、掛け算やら足し算やらをすればいいだけ、ということも、さっき見たとおりです。

文字列を数字になおす方法は、intという関数を使います。これの使い方を、対話シェルでの実験結果を含めて下に示しておきます。

```
>>> "200" + 10        ←数字に見えても、文字列である限りは計算に使えない
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> int("200")        ←intという関数で処理すると、ダブルクォーテーション記号で囲まない数字が表れた
200
>>> int("200") + 10    ←これだと計算につかえる。「数」に変換されたから
210
```

なるほど、intというものをつかうのね。ということで、まずは安直に下のようなことを試してみます。

```
>>> int("17:30")   ←こいつを「数字」とみなしたいんだけど…
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '17:30'
```

怒られました。コロン記号が含まれていたりすると、まともな数字には見えないからダメみたいです。

"17:30"を、五文字からなる文字列としてよく見てみましょう。これの1文字目から2文字目、4文字目から5文字目をそれぞれ部分文字列として抜き出したら、それなら数字として扱うことが可能そうではないですか。具体的には、[0:2]と、[3:5]です。対話シェルから試してみましょう。

```
>>> s = "17:30"  ←扱いやすいように、一旦変数にこの文字列を入れて…
>>> s[0:2]   ←部分文字列を出す
'17'
>>> int(s[0:2])   ←これならintで数字に変換できる
17
>>> int(s[3:5])   ←同様に、分のほうも処理できる
30
```

こういう方法以外にもいろいろやりかたはありますが、まずは使えそうなワザから試してみようということで、この線で行きましょう。

ここまでで、細かい技術的要素はすべて説明しおわりました。今までの知識を結集して、下の練習問題をやってみてください。

練習問題01

【練習問題：01】今から示す退社時間ファイル(leavetime.txt)をすべて読み込んで、残業時間（と分）の合計を算出せよ。ただし、残業開始時間は午後5時20分とする。 スクリプト名はなんでもよいが、参考までに、後の模範解答では zangyo_sum.py などとする予定である。 結果が10時間15分なら、10 hours 15 minutes などと表示すればよい。無理に日本語を出さなくてよい。

　leavetime.txt ←右クリックから「リンク先を保存」「対象を保存」などを選んで保存してください。