Python 最適化第 3 回：最適化によるシフト作成

従業員の都合を考慮して，時間ごとに必要な労働力を満たすデイリーシフトスケジュールを，最適化問題を解くことで作成する方法を学習する．

1. シフト作成問題（その１）：基本モデル

1.1 対象の問題（シート「シフト 1」参照）

ファミレスで働くアルバイトに対するデイリーシフト表を作成する．シート「シフト 1」内から必要な情報を読み取り，表「立案スケジュール」内に決定したシフト内容を書き込む．書き込む内容は，K（キッチン），F（フロア），D（デリバリー），未入力（割付仕事なし）のどれか．

1.2 問題の概要

- 期間は 9:00～22:00 までの 13 時間．30 分単位でメンバーが行う仕事を決定する．
- 30 分の単位をスロットと呼ぶ．期間は 13 時間なので，26 スロット．
- 1 日に勤務可能な時間は 9 時間（18 スロット）まで
- 仕事は K（キッチン），F（フロア），D（デリバリー）のいずれか．どの仕事もしないことも可能．
- スロットごとに各仕事に必要な人数が決まっている．
- 対象の日に勤務可能なメンバーは 8 名．ただし，メンバーによって，働けるスロットはまちまち．担当できる仕事もまちまち．
- メンバーの能力とスロット（勤務時間帯）によって，支払うアルバイト代が異なる
- 条件を満たし，支払うアルバイト額を最小にするシフトスケジュールを作成したい

まずは手始めに，どのスロットでも働けて，どの仕事も担当できる，という条件で問題を解こう．

1.3 シート「シフト」の説明

- 立案スケジュール：
  - セル B3:AA10：　最適化で決定した仕事（K，F，D）を記入する枠
  - AB 列：　決定した勤務時間．9 時間を超えるとフォントが赤になる
  - 11～13 行：　スロットごとの K・F・D 担当人数．不足だとフォントが赤に．
- 支払額（AB13）：　アルバイト代の総額．この金額を最小化したい
- 単価（A16:AA24）：　各メンバーを各スロットで勤務させたときに必要な費用．メンバーによって，スロットによって，費用が異なる．
- 必要工数（A27:AA30）：　各スロットにおいて各仕事に必要な人数

1.4 最適化における集合・定数

- 最適化では，仕事 K，F，D をそれぞれ仕事 1，2，3 と呼ぶ
- $c_{i,t}$　　メンバーi がスロット t で勤務するときに必要なアルバイト代　（単価）
- $d_{j,t}$　　スロット t で必要な仕事 j をするメンバーの人数　（必要工数）

1.5 決定変数

- $x_{i,j,t}$　　メンバーi が仕事j をスロットt にするなら 1，そうでないなら 0.

  例) メンバー1 が仕事 2 をスロット 3 にする場合，$x_{1,2,3} = 1$

  メンバー3 がスロット 4 には何の仕事もしない場合，$x_{3,1,4} = x_{3,2,4} = x_{3,3,4} = 0$

1.6 制約条件

- 各スロットにおいて，各仕事に必要な人数以上のメンバーを勤務させる
- どのメンバーも，勤務時間は 9 時間（18 スロット）まで
- 各メンバーは，1 スロット内で行う仕事は 1 種類まで
- （決定変数は 0 か 1 しか取らない）

1.7 目的関数

メンバーに支払うアルバイト代の合計金額の最小化

1.8 定式化

最？化　　$\sum_{?=?}^{?}\sum_{?=?}^{?}\sum_{?=?}^{?} ?$　　（メンバーに支払う費用の総和）

条件：

$\sum_{?=?}^{?} ? ? ? \quad (j = 1,2,3, \quad t = 1,2,\dots,26)$　　（スロットtで仕事j に必要な人数）

$\sum_{?=?}^{?}\sum_{?=?}^{?} ? ? ? \quad (i = 1,2,\dots,8)$　　（メンバーi の勤務時間上限）

$\sum_{?=?}^{?} ? ? ? \quad (i = 1,2,\dots,8, \quad t = 1,2,\dots,26)$

（メンバーiのスロットtで行う仕事は 1 種類まで）

$$x_{i,j,t} = 0 \; or \; 1 \quad (i = 1,2,\dots,8, \quad j = 1,2,3, \quad t = 1,2,\dots,26)$$

研究室配属問題（その３）のプログラムを部分的にコピペして，以降の解説を読みながら，今回の目的のプログラムを作成するために追加・変更しよう．

1.9 Python プログラム（途中から）：空問題の作成

```
model = LpProblem('Shift', sense=Lp???)
```

- 問題名は任意だが，ここでは Shift とした．最大化か最小化か，いずれかを設定する．

1.10　Python プログラム（途中から）：制約条件の追加・目的関数の設定

```
for ??? in ???:
    for ??? in ???:
        mode += ???                #仕事に必要な人数
for ??? in ???:
    model += ???                   #勤務時間上限
for ??? in ???:
    for ??? in ???:
        model += ???               #仕事は 1 種類まで
model += lpSum(??? for i in I for j in J for t in T)
```

- 最初から全ての制約条件をプログラム化するのが難しい場合は，条件を 1 種類ずつ追加して実行するのもいい方法.
- 【余裕のある人向け】上述のプログラム構造は理解しやすさを重視したもの．複数の制約条件を同じ for 文の中に入れることで，for 文の個数を減少させることも可能.

1.11　Python プログラム（途中から）：最適化の結果出力

```
J_Name = {1:'K', 2:'F', 3:'D'}
for i in I:
    for j in J:
        for t in T:
            if value(x[i, j, t]) > 0.01:
                sheet.cell(row=2+???, column=1+???).value = J_Name[???]
```

- シート「シフト」のセル範囲 B3:AA10 に，割り当てる仕事名（K か F か D）を出力する．最適化では仕事名の代わりに通し番号 1, 2, 3 を使っているので，通し番号から元の仕事名に戻すために，辞書 J_Name を定義している
- 【余裕のある人向け】辞書ではなくリスト J_Name = [‘’, ‘K’, ‘F’, ‘D’]でも実現可能

◎ 課題 1

シフト作成問題（その 1）を最適化問題として定式化しなさい．その後，最適解を求めるプログラムを作成しなさい.

Excel ファイル**最適化 3.xlsx** のシート「シフト 1」から必要なデータを読み込み，求めた結果を同シート内に出力して，ファイル**最適化 3-1.xlsx** として保存すること．（最適値：40050）

プログラムを実行したら，結果を出力した Excel ファイルを確認して，制約条件が満たされているか確認しよう．（以降の課題でも同様）

2. シフト作成問題（その２）：都合と担当可能性を考慮

残りの条件である都合と担当可能性を追加しよう．

2.1 シート「シフト＋」の説明（シート「シフト」と異なる部分のみ）

- 都合（A33:AA41）：　　各メンバーが働けるスロットに 1，働けないスロットに 0
- 担当可能（A44:D52）：　　各メンバーがその仕事を担当できる場合に 1，できない場合に 0

2.2 最適化における集合・定数（追加分）

- $a_{i,t}$　　メンバーi がスロット t で勤務可能なら 1，不可能なら 0　（都合）
- $b_{i,j}$　　メンバーi が仕事 j を担当可能なら 1，不可能なら 0　（担当可能）

2.3 決定変数・目的関数

前の問題と同じ

2.4 制約条件（追加分）

- メンバーが担当不可能な仕事を割り当てない
- メンバーが勤務不可能なスロットには，仕事を割り当てない

2.5 追加分の制約条件の定式化

$$？ = 0 \quad （？ = 0 \text{ のとき}） \quad (i = 1, 2, \dots, 8, \quad j = 1, 2, 3, \quad t = 1, 2, \dots, 26)$$

（メンバーi がスロット t で勤務不可能なら，どの仕事 j も割当しない）

$$？ = 0 \quad （？ = 0 \text{ のとき}） \quad (i = 1, 2, \dots, 8, \quad j = 1, 2, 3, \quad t = 1, 2, \dots, 26)$$

（メンバーi が仕事 j を担当できないなら，どのスロット t でも割当しない）

これらの制約条件は，$a_{i,t}$ や $b_{i,j}$ という i, j, t という添え字を持つ定数がある条件を満たす場合のみ設定する．$a_{i,t}$ や $b_{i,j}$ 等は Excel ファイルから読み込む定数であり，最適化問題を解く前に定まっている定数である．したがって，ここでは「（$？ = 0$ のとき）」のように，定式化の中に条件文を入れることができる．

一方，「$x_{i,j,t} = 0$ のとき」のように，決定変数がある条件を満たす場合のみ，制約条件を課すことはできない．なぜなら，決定変数の値は最適化問題を解かないと確定しないためである．決定変数の値に関して「・・・のとき」という制約条件は，表現を変更することで「・・・のとき」を含まない制約条件で表せることが多い（詳細は後述）．

2.6 Python プログラム（途中から）：制約条件の追加

```
for ??? in ???:
    for ??? in ???:
        if ??? == 0:      #勤務不可能な時間帯なら
            for ??? in ???:
                model += ???          #割り当てない
for ??? in ???:
    for ??? in ???:
        if ??? == 0:      #仕事を担当できないなら
            for ??? in ???:
                model += ???          #割り当てない
```

- 上のプログラム内の「担当不可」や「割当不可」は，$a_{i,t}$や$b_{i,j}$という定数の値で判断している．定数は最適化を実行する前に値が確定しているので，それらの値を使った if 文を使って，「担当不可なら，割り当てない」等を表現できる．
- 【余裕のある人向け】$x_{i,j,t}$ が 0 か 1 しか取らないことから，例えば各 t に対して$x_{i,j,t} = 0$とする代わりに，$\sum_t x_{i,j,t} = 0$ とすることも可能．

◎ 課題 2

シフト作成問題（その 2）を最適化問題として定式化しなさい．その後，最適解を求めるプログラムを作成しなさい．

Excel ファイル最適化 3-1.xlsx のシート「シフト 2」から必要なデータを読み込み，求めた結果を同シート内に出力して，ファイル最適化 3-2.xlsx として保存すること．（最適値：41615）

3. シフト作成問題（その３）：最低勤務時間

シフト作成問題（その 2）では，短時間しか勤務しないメンバーが存在し，「出勤するならせめて 4 時間は働かせてほしい」という要望があった．そこで，出勤するなら，4 時間（8 スロット）以上は勤務するような制約条件を追加する．

3.1 追加する決定変数

「出勤するなら」を実現するために，次の決定変数を追加する．

- y_i　　メンバーi が 1 スロットでも仕事をするなら 1，そうでないなら 0

  例) メンバー1 が対象の日に 1 スロットでも仕事をする場合，$y_1 = 1$

  メンバー3 が対象の日に 1 スロットも仕事をしない場合，$y_3 = 0$

3.2 追加する制約条件

- 1 スロットでも仕事を割り付けるなら，4 時間（8 スロット）以上の仕事を割り付ける．

3.3 定式化の考え方

y_iの値によって，出勤時間に関する制約条件がどのように変わるか考えよう．

- ✧ $y_i = 1$ の場合（1 スロットでも勤務する場合）

  この場合，勤務時間は 4 時間（8 スロット）以上，9 時間（18 スロット）以下．よって，従業員 i の勤務時間に関する制約は次のように書ける．

$$8 \leq \sum_j \sum_t x_{i,j,t} \leq 18$$

- ✧ $y_i = 0$ の場合（出勤しない場合）

  この場合，勤務時間は 0 時間なので，従業員 i の勤務時間に関する制約は以下の通り．

$$\sum_j \sum_t x_{i,j,t} = 0$$

上の結果を見ると場合分けになっているので，「課題 2 のように，if 文を使って場合ごとの制約条件を立てればよい」と考えるかもしれないが，それは**間違い**である．

課題 2 では，$a_{i,t}$や$b_{i,j}$という定数の値によって，制約条件の立て方を変えていた．定数は最適化問題を解く前に値が定まっているので，if 文を使って条件に応じた制約条件を立てることができる．

一方，今回の場合分けは y_iという決定変数の値によって行われている．決定変数の値は最適化によって求めるものだから，最適化問題を解く前に値は定まっておらず，最適化モデルを作成する Python プログラム内で if 文を使って条件に応じた制約条件を立てることはできない．

そこで，上で表した$y_i = 1$ の場合と$y_i = 0$ の場合を統合して，両方の場合を表す式を作成しよう．

$y_i = 0$ の場合の表現を変えると，次のように書くことができる．

- ✧ $y_i = 1$ の場合（1 スロットでも勤務する場合）

$$8 \leq \sum_j \sum_t x_{ijt} \leq 18$$

- ✧ $y_i = 0$ の場合（出勤しない場合）

$$0 \leq \sum_j \sum_t x_{ijt} \leq 0$$

このように，両方の場合を定数が異なるだけの同じ形式で表すことができた．あとは

- ✧ 左辺：　$y_i = 1$のとき 8，$y_i = 0$のとき 0
- ✧ 右辺：　$y_i = 1$のとき 18，$y_i = 0$のとき 0

となるように左辺と右辺の定数をy_iの式で表せば，$y_i = 1$ の場合と$y_i = 0$ の両方の場合を表す式となる．

この新しい勤務時間に関する制約条件は，既に存在した制約条件「勤務時間は 9 時間以内」を包含するので，「勤務時間は 9 時間以内」の制約式は定式化から省くことが可能（残っていても問題は

ないが，最適化ソルバーの処理時間が長くなることが多い（今回のサイズなら問題なし)).

【注意点】（余裕のある人向け）

今回追加する条件は「出勤するなら 4 時間以上勤務する」という勤務時間に関する下限の制約なので，この条件には上限の設定は不要なのではないか？と疑問に感じるかもしれない．

仮に上限の制約はこれまで通り 9 時間まで，つまり勤務スロット数≦18 という式にして，下限の制約は

- ✧ $y_i = 1$ の場合（1 スロットでも勤務する場合），8≦勤務スロット数
- ✧ $y_i = 0$ の場合（出勤しない場合），0≦勤務スロット数

としたとしよう．上限の制約には，y_iが含まれないことに注意しよう．

このように表現して最適化を実行すると，4 時間未満の勤務を立案することがある．例えば，本来は許されていない勤務スロット数＝2（勤務時間 1 時間）となるように $x_{i,j,k}$ の値を定めても，$y_i = 0$ とすれば「0≦勤務スロット数」となり，勤務時間の下限の制約を満たしてしまう．言い換えれば，本当なら勤務スロット数＝2 であれば，$y_i = 1$ になってほしいのに，上のような表現では，$y_i = 1$ にするための制約式が存在しない．

今回のように，

- ✧ $y_i = 1$ の場合（1 スロットでも勤務する場合），8≦勤務スロット数≦18
- ✧ $y_i = 0$ の場合（出勤しない場合），0≦勤務スロット数≦0

とすれば，勤務スロット数＝2 は，勤務スロット数に関するどちらの条件も満たさないので，勤務スロット数＝2 にすることはできない．立案可能なのは，勤務スロット数＝0，または 8～18 になるときのみである．

3.4 Python プログラム（途中から）：決定変数の追加

```
y = {}
for ??? in ???:
    y[???] = LpVariable(f'y{???}', cat=LpBinary)
```

- 制約条件を追加する前に，決定変数 y_i を作成する．

3.5 Python プログラム（途中から）：制約条件の追加

```
for ??? in ???:
    model += lpSum(x[i,j,t] for j in J for t in T) >= ???        #勤務時間の下限
    model += lpSum(x[i,j,t] for j in J for t in T) <=???        #勤務時間の上限
```

- 定式化では，勤務時間の上限と下限を 2 つの≦を含む式で表現したが，PuLP で追加できる制約条件に含む等号・不等号は 1 つまでなので，上限に関する制約と下限に関する制約を別個に追加する．
- 定式化では Σ2 つ必要であったが，PuLP では lpSum 1 つで表現可能．
- 課題 2 までに作成した「勤務時間は 18 スロット（9 時間）まで」のプログラムは削除してもよい．

◎ 課題 3

シフト作成問題（その３）を最適化問題として定式化しなさい．その後，最適解を求めるプログラムを作成しなさい．

Excel ファイル最適化 3-2.xlsx のシート「シフト 3」から必要なデータを読み込み，求めた結果を同シート内に出力して，ファイル最適化 3-3.xlsx として保存すること．（最適値：43215）

Excel を開いて，勤務している人が最低 4 時間（8 スロット）以上勤務しているか確認しよう．

4. シフト作成問題（その４）：飛び石禁止

シフト作成問題（その３）では，勤務するスロットの間に空きスロットがあった．「給料を支払うのは働いた時間の分のみ．空き時間の分は給料を支払わない」とのマネージャーのセリフを聞いたメンバーは立腹して，「じゃあ，空き時間が入れるようなシフトを作るな！」とマネージャーに詰め寄った．そこでマネージャーは，勤務するスロットの間に空きスロットがないようなシフトを作成しなければならなくなった...

4.1 追加する決定変数

その日の仕事開始スロットと仕事終了スロットを表す決定変数を追加する．

- $s_{i,t}$　　メンバーi がスロット t でその日仕事を開始したのなら 1，そうでないなら 0
- $e_{i,t}$　　メンバーi がスロット t までその日仕事をしたのなら 1，そうでないなら 0

例）

スロット t	1	2	3	4	5	6	7	8	9
メンバーi の仕事	---	---	---	F	F	K	K	---	---
$s_{i,t}$	0	0	0	1	0	0	0	0	0
$e_{i,t}$	0	0	0	0	0	0	1	0	0

4.2 追加する制約条件

(1) その日に出勤するなら，開始スロットと終了スロットを定める．

⇒ メンバーi が出勤する日は，あるスロット t のみ $s_{i,t}=1$，他のスロットは $s_{i,t}=0$.

同様に，あるスロット t のみ $e_{i,t}=1$，他のスロットは $e_{i,t}=0$.

⇒ メンバーi が出勤しない日は，全スロット t で $s_{i,t}=0$，全スロット t で $e_{i,t}=0$.

(2) 開始スロットが終了スロットより遅くならない．終了スロットの方が早くならない．

(3) 仕事をするスロットは，開始スロット～終了スロットの間のみに限定する．

⇒ 開始スロットより前に仕事をしない．終了スロットより後に仕事をしない．

4.3 定式化の考え方（その１）：出勤するなら，$s_{i,t}$ と $e_{i,t}$ を定める

メンバーi が出勤するときは $y_i = 1$，出勤しないときは $y_i = 0$ である．課題 3 と同様，他の決定変数の値によって場合分けをする必要があるので，Python プログラム内で if 文を使うことはできない．開始スロット $s_{i,t}$ について，場合分けして考えてみよう．

✧ $y_i = 1$ の場合（1 スロットでも勤務する場合）

この場合，どこかで開始する，つまりあるスロット t のみ $s_{i,t} = 1$，他のスロットは $s_{i,t} = 0$ と必要がある．（$s_{i,t}$ に関する制約はどのように書けるかな？）

✧ $y_i = 0$ の場合（出勤しない場合）

この場合，全スロット t で $s_{i,t} = 0$ である．（$s_{i,t}$ に関する制約はどのように書けるかな？）

この 2 つの制約式は同じ形式で定数の値が異なるだけなので，y_iを用いて定数を表現すればよい．終了スロット $e_{i,t}$ についても同様に考える．

4.4 定式化の考え方（その２）: $s_{i,t}$ と $e_{i,t}$ の前後関係

あるスロット t' において，メンバーi がすでに仕事が開始したのか，メンバーi がすでに仕事を終了したのか，を確認することにする[1]．「すでに」を確認するには，仕事の開始スロットであるスロット 1 からスロット t' まで，つまり $t = 1, 2, \ldots, t'$ を調べればよい．

例）

スロット t'	1	2	3	4	5	6	7	8	9
$s_{i,t'}$	0	0	0	1	0	0	0	0	0
$\sum_{t=1}^{t'} s_{i,t}$	0	0	0	1	1	1	1	1	1
$e_{i,t'}$	0	0	0	0	0	0	1	0	0
$\sum_{t=1}^{t'} e_{i,t}$	0	0	0	0	0	0	1	1	1

$\sum_{t=1}^{t'} s_{i,t} = 1$なら，スロット t' かそれ以前に仕事を開始していること，$\sum_{t=1}^{t'} e_{i,t} = 1$なら，スロット t' かそれ以前に仕事を開始していることを表す．開始スロット≦終了スロットが成立すれば，どのスロット t'においても，

$$\sum_{t=1}^{t'} s_{i,t} \geq \sum_{t=1}^{t'} e_{i,t}$$

が成立することが分かる．逆にどのスロット t'においてもこの式を満たすならば，開始スロット≦終了スロットを満たすことができる．

4.5 定式化の考え方（その３）: 仕事をするスロットは，開始スロットと終了スロットの間

メンバーi が出勤してあるスロット t' で仕事をする場合，開始スロット≦スロット t'≦終了スロット，の大小関係が成立する必要がある．次ページに示す図を用いて考える．

[1] $s_{i,t}$ と $e_{i,t}$そのものを「スロット t で既に仕事を開始／終了したか」を表す変数にするモデル化も考えられる．また$e_{i,t}$でのスロット t を仕事を行う最終スロットとみるか，仕事を終了した最初のスロットとみるかによっても，定式化が少し異なる．このように同じ問題に対しても，モデル化や定式化は様々であり，その方法によって計算時間が異なることも多い．

図(a)の場合，スロット t' は開始スロットと終了スロットの間になので，仕事を行う必要がある．言い換えると，スロット 1～スロット t'の範囲（図の青い中括弧）に開始スロットが存在し，スロット t' から最終スロットの範囲（図の緑色の中括弧）に終了スロットが存在しなければならない．『スロット 1～スロット t'の範囲（図の青い中括弧）に開始スロットが存在』という条件を式で表すと，

$$\sum_{t=1}^{t'} s_{i,t} = 1$$

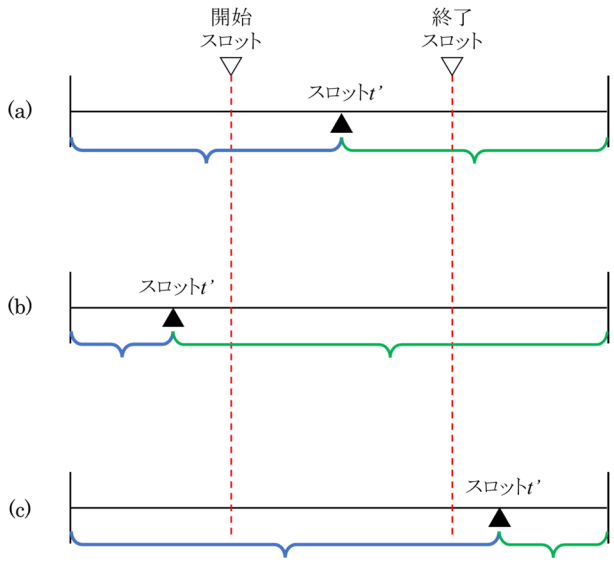
となり，『スロット t' から最終スロットの範囲（図の緑色の中括弧）に終了スロットが存在』という条件を式で表すと，

$$\sum_{t=t'}^{26} e_{i,t} = 1$$

となる．よって，『メンバーi が出勤して，スロット t'が開始スロットと終了スロットの間なら，メンバーi はスロット t'で仕事を行う』は，

$$y_i = 1, \quad \sum_{t=1}^{t'} s_{i,t} = 1, \quad \sum_{t=t'}^{26} e_{i,t} = 1 \quad \Rightarrow \quad \sum_{j=1}^{3} x_{i,j,t'} = 1$$

と表現できる．



次に図(b)の場合は，『メンバーi が出勤して，スロット t'が開始スロットより前なら，メンバーi はスロット t'で仕事を行ってはならない』を表している．4.4 節より，終了スロットは開始スロットより後なので，終了スロットは必ずスロット t'より後になる．よって，この場合の条件は，

$$y_i = 1, \quad \sum_{t=1}^{t'} s_{i,t} = 0, \quad \sum_{t=t'}^{26} e_{i,t} = 1 \quad \Rightarrow \quad \sum_{j=1}^{3} x_{i,j,t'} = 0$$

と表現できる．

同様に図(c)の場合は，『メンバーi が出勤して，スロット t'が終了スロットより後なら，メンバーi はスロット t'で仕事を行ってはならない』なので，この場合の条件は，

$$y_i = 1, \quad \sum_{t=1}^{t'} s_{i,t} = 1, \quad \sum_{t=t'}^{26} e_{i,t} = 0 \quad \Rightarrow \quad \sum_{j=1}^{3} x_{i,j,t'} = 0$$

と表現できる．

最後に，メンバーiが出勤しない場合は，4.3 節より全ての t に対して $s_{i,t} = e_{i,t} = 0$ となるので，

$$y_i = 0, \quad \sum_{t=1}^{t'} s_{i,t} = 0, \quad \sum_{t=t'}^{26} e_{i,t} = 0 \quad \Rightarrow \quad \sum_{j=1}^{3} x_{i,j,t'} = 0$$

と表現できる．

以上のすべての場合において，$\sum_{j=1}^{3} x_{i,j,t'}$ の値が正しく設定できるように制約条件を作成したい．4 つの場合における式をよーく見て考えると，次の式が作成できる[2]．

$$\sum_{t=1}^{t'} s_{i,t} + \sum_{t=t'}^{26} e_{i,t} = \sum_{j=1}^{3} x_{i,j,t'} + y_i$$

4.6 Python プログラム（途中から）：決定変数の追加

```
s = {}
e = {}
for ??? in ???:
    for ??? in ???:
        s[???] = LpVariable(f's{???},{???}', cat=LpBinary)
        e[???] = LpVariable(f'e{???},{???}', cat=LpBinary)
```

- 制約条件を追加する前に，決定変数 $s_{i,t}$，$e_{i,t}$を作成する．

4.7 Python プログラム（途中から）：制約条件の追加

```
for ??? in ???:
    model += ??? ??? ???          #【条件その 1】：本日出勤なら，開始スロットが必要
    model += ??? ??? ???          #【条件その 1】：本日出勤なら，終了スロットが必要
    for tt in T:
        model += ??? >= ???       #【条件その 2】：開始スロット≦終了スロット
        model += ??? == ???
                                  #【条件その 3】：仕事スロットは，開始スロットと終了スロットの間
```

- プログラムでは，t'の代わりに tt を用いている．
- 【条件その 2】では，Σの対象がスロット 1 からスロット t'までとなる．例えば $\sum_{t=1}^{t'} s_{i,t}$ は

[2] $y_i\left(\sum_{t=1}^{t'} s_{i,t}\right)\left(\sum_{t=t'}^{26} e_{i,t}\right) = \left(\sum_{j=1}^{3} x_{i,j,t'}\right)$という式を思いついた人もいるかもしれないが，この式だと決定変数に対して線形（一次式）ではないので，今回用いている最適化ソルバーでは求解できない．

次のように書ける．

```
lpSum(s[i,t] for t in range(1, tt+1))
```

- 【条件その 3】での $e_{i,t}$ は，Σの対象がスロット t'から最終スロット（スロット 26）までとなるので，次のように書ける．

```
lpSum(e[i,t] for t in range(tt, 26+1))
```

- 【条件その 3】を満たせば，開始スロットから終了スロットの間のスロットで $\sum_{j=1}^{3} x_{i,j,t'} = 1$ にしかならないので，これまで存在した「1 スロットで行う仕事は 1 種類まで」の制約条件である以下の部分は削除可能．残しておいても問題はない．

```
for i in I:
    for t in T:
        model += lpSum(x[i,j,t] for j in J) <= 1
```

◎ 課題 4

シフト作成問題（その４）を最適化問題として定式化しなさい．その後，最適解を求めるプログラムを作成しなさい．

Excel ファイル**最適化 3-3.xlsx** のシート「シフト 4」から必要なデータを読み込み，求めた結果を同シート内に出力して，ファイル**最適化 3-4.xlsx** として保存すること．（最適値：49835）

飛び石勤務がなくなったか，実行結果を見て確認しよう．

【注意】これまでの問題より，計算時間がかかります．

【注意】課題 1～3 までの定式化で，厳密には間違っている定式化でも正しく最適解を求めることができていた，というケースが存在します．その場合，この問題で「最適解が見つかりません」が表示されることになります．

目的関数は費用最小化なので，シフトではなるべく勤務時間を減らすようにします．ところが，今回追加した制約「飛び石の禁止」により，本来必要ないのに勤務してもらう必要性が生じることがあります．このことに留意して定式化を修正すれば，「最適解が見つかりません」の問題は解決するでしょう．

5. シフト作成問題（その５）：終日同一仕事（挑戦問題）

シフト作成問題（その４）でようやく飛び石勤務をなくしたシフト表を作成できたマネージャーがほっとするのもつかの間，シフト表を見たメンバーから「担当する仕事が時間帯ごとにコロコロ変わったら，いつ何の仕事をすればよいか覚えきれない！」と文句を言われた．

そこで，1 日に担当する仕事は 1 種類までに限定してシフトを作成できるか，試してみることにした．

5.1 追加する決定変数

ある時点でその日（1 スロットでも）仕事をしたかどうかを表す決定変数を追加する．

- $w_{i,j}$　　メンバーi が仕事 j を 1 スロットでも担当するなら 1，そうでないなら 0

5.2 追加する制約条件

- 1 日で行う仕事の種類は 1 種類まで

5.3 定式化の考え方（その１）: 出勤するなら，1 日に担当する仕事の数は 1 つ

研究室配属問題にて，「1 サイクルに配属する研究室は 1 つ」と同様．決定変数 w_{ij} で表す．出勤しないときは担当する仕事はありません．

5.4 定式化の考え方（その２）: 決定変数 w_{ij} による影響

5.3 の制約条件だけだと，他の決定変数に何の影響も与えません．w_{ij}の値に応じて，他の決定変数に制約が加わるように，制約条件を追加，または既に存在しているある制約条件を変更する必要があります．

◎ 課題 5

シフト作成問題（その５）を最適化問題として定式化しなさい．その後，最適解を求めるプログラムを作成しなさい．

Excel ファイル最適化 3-4.xlsx のシート「シフト 5」から必要なデータを読み込み，求めた結果を同シート内に出力して，ファイル最適化 3-5.xlsx として保存すること．（最適値：51550）

勤務しているメンバーが担当する仕事が 1 種類かどうか確認しよう．

6. シフト作成問題：今後の展望

シフト作成はアルバイトが勤務する多くの業界・店舗で行われる．今回の内容では，多くの業界・店舗で考慮することが多い条件を取り扱った．各業界・店舗ごとに異なる考慮条件も存在し，それらによってシフト作成の難易度が異なる．各業界・店舗に対して，すぐに利用できる実用的な個別のシフト作成システムを構築するのは，重要な研究である．

一方，特定の業界・店舗を対象とするのではなく，広範囲の業界・店舗を対象とした汎用的なシフト作成システムを構築することも重要である．そのシステムは多くのシフト作成で考慮される条件を基盤として備え，シフトを作成する人が自分の業界・店舗に必要な考慮条件をシステムに対して指定・設定してカスタマイズすることで，プログラムコードを作成・変更することなく，シフト作成者の要望に応じたシフトを出力する．カスタマイズがより簡単で，実行時間がより短く，質がより良いシフトを作成できる汎用システムの開発を行うのも重要な研究である．

以上

【ヒント】（課題が解けず，ヒントが欲しい人向け）

◎ 課題 1

この問題は，前回扱った研究室配属と構造が似ています．研究室配属問題では，どの学生 i をどの研究室 j にどのサイクル t に配属するのかを決定します．一方，シフト作成問題では，どのメンバー i をどの仕事 j にどのスロット t で担当させるのかを決定します．前回の研究室配属問題で作成した定式化やプログラムを参照して，問題を解きましょう．

プログラムでは，J_Name = {1:'K', 2:'F', 3:'D'}と設定しています．言い換えると，J_Name[1] = ‘K’，J_Name[2] = ‘F’，J_Name[3] = ‘D’，です．if value(x[i, j, t]) > 0.01: が成立するのは，最適化の結果，x[i, j, t] = 1 となったときです．つまり，メンバー i に仕事 j をスロット t で担当させるときです．Excel に j の値そのものを表示しても何の仕事か分かりにくいので，ここでは代わりに J_Name[j] を表示します．Excel のシートをよく見て，正しい位置のセルに出力されるようにプログラムを完成させましょう．

◎ 課題 2

課題 1 からの変更点は，定数 $a_{i,t}$ や $b_{i,j}$ を Excel から読み込むことと，2.6 節で記した制約条件を追加することだけです．正しく動かない場合は print(a) などを使って，Excel 内のデータがプログラムで正しく定数として値を格納できているか確認しましょう．

◎ 課題 3

勤務スロット数の上限・下限に関する定式化さえできれば，定式は課題 2 とほぼ同じです．決定変数が増えているので，忘れずに追加しましょう．

プログラムも資料の**???**を埋めれば完成です．出力ファイル名の変更もお忘れなく．

◎ 課題 4

この課題の内容はかなり難易度が高いため，資料をきちんと読めば解けるようにしています．変数も多く，Σ も多くて複雑ですが，あきらめずにじっくり読んで理解しましょう．

飛び石勤務を禁止するために，仕事に必要な人数より多くの従業員を配置しなければならないスロットが生じます．そのため，各スロット t で各仕事 j に割り当てる人数は，ちょうど $d_{j,t}$ 人ではなく，$d_{j,t}$ 人で以上にする必要があります．課題 3 までは運良く「ちょうど $d_{j,t}$ 人」としていても最適解を求めることができましたが，この問題では不可能になります．

【大ヒント】（【ヒント】を見ても，どうしても課題が解けない人向け！）

◎ 課題 1

制約条件「仕事に必要な人数」は，仕事 j とスロット t の各ペアに対して 1 つの制約条件を設定します．したがって，プログラムでは，j と t の 2 重ループになります．決定変数 $x_{i,j,t}$ のうち，ループによって j と t が定まるので，残るは i です．「必要な人数」は i に関する Σ を使って表せます．

同様に，制約条件「勤務時間上限」は各 i に対する制約なので，i のループ内に j に関する Σ と t に関する Σ を使って表される条件式が作成されます．

最後に，制約条件「仕事は 1 種類まで」は各 i と t に対する制約なので，i と t の 2 重ループ内に j に関する Σ を使って表される条件式が作成されます．

◎ 課題 2

「割り当てない」は決定変数の値を 0 に指定するだけです．例えば，「メンバーi に仕事 j をスロット t では割り当てない」なら，$x_{i,j,t} = 0$ です．

◎ 課題 3

勤務時間の下限は，$y_i = 1$のとき 8，$y_i = 0$のとき 0 です．y_iの値によって下限が異なるので，下限は y_i に関する数式にする必要があります．

仮に求める数式を $2y_i$ としてみましょう．すると，$2y_i$ は $y_i = 1$のとき 2，$y_i = 0$のとき 0 です．

◎ 課題 4

【条件その 1】の開始スロットに関する制約の左辺は，$s_{i,t}$ の合計になります．右辺は y_i の値によって異なるため，y_i に関する数式になります．終了スロットについても同様です．

【条件その 2】の制約条件式は 4.4 節に記されています．Σ の表現方法は 4.7 節で述べています．

【条件その 3】の制約条件式も【その 2】と同様にほぼ資料に記されています．$x_{i,j,t}$ではなく$x_{i,j,t'}$ である点に注意しましょう．

7.

8. シフト作成問題（その 6）: 休憩の追加（挑戦問題）

シフト作成問題（その５）で，一人 1 種類の仕事割り当てでもシフト表を作成できたマネージャーがほっとするのもつかの間，複数のファミレス店舗を管理するエリアマネージャーから「アルバイトは 4 時間連続勤務させたら，そのあと 30 分は休憩取らせてね.」と指示された．

そこで，指示通りに 4 時間連続勤務した直後に 30 分の休憩を入れるシフトを作成できるか，試してみることにした．

8.1 追加する設定

休憩時間も給料は支払うので，休憩も 1 つの仕事として扱おう．仕事名を K，F，D に加えて「－」としよう．最適化では仕事は番号で扱っていたので，休憩「－」の番号を 4 としよう．

8.2 追加する決定変数

直前までに 4 時間連続で勤務して，その時点で休憩をとる必要があるかどうかを表す決定変数を追加する．ただし，「4 時間連続」の対象となる仕事の中に，休憩「－」は含まない．

- $v_{i,t}$　　メンバーi がスロット t で休憩を取らなければならないなら 1，そうでないなら 0

8.3 追加する制約条件

- 休憩「－」以外の仕事を 4 時間連続勤務したら，直後に休憩「－」を入れる

8.4 定式化の考え方（その１）: 1 日に担当する仕事の数は 1 つまで

研究室配属問題にて，「1 サイクルに配属する研究室は 1 つ」と同様．決定変数 w_{ij} で表す．1 日まったく勤務しないメンバーも存在する可能性もあるので，「1 つまで」となることに注意．

8.5 定式化の考え方（その２）: 決定変数 w_{ij} の設定

今回扱った定式化を復習すれば，同様の方法で実現可能．

w_{ij}に関する制約条件を加えることで，既に存在しているある制約条件が不要になるので，削除してもよい．

◎ 課題 5

シフト作成問題（その５）を最適化問題として定式化しなさい．その後，最適解を求めるプログラムを作成しなさい．

Excel ファイル kadai20-4-4.xlsx のシート「シフト 5」から必要なデータを読み込み，求めた結果を同シート内に出力して，ファイル kadai20-4-5.xlsx として保存すること．(最適値 : 50942.5)

勤務しているメンバーが担当する仕事が 1 種類かどうか確認しよう．