Python3 制約 プログラミングマニュアル

Mar.17.2020 菅原システムズ

はじめに

Python3を用いた制約の記述の仕方について述べます。

Python3 を用いた制約記述とは

殆どは、GUIで記述可能です。必要がなければGUIで記述してください。
Python3言語制約を使うのは、次の場合に限られます。

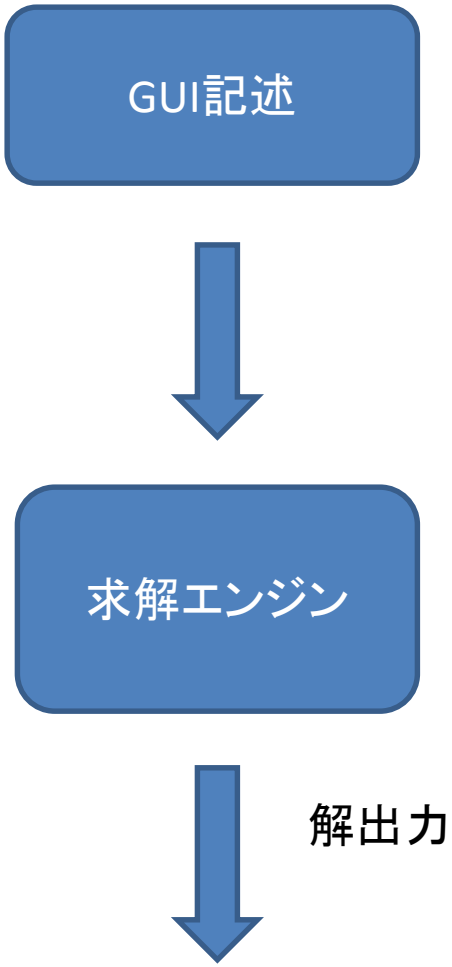
■GUIでは、記述不能な制約がある

■GUIの制約のオンオフを動的に行いたい

■メンテナンス上の理由。Python3で動的記述した方がメンテナンス性が向上する場合

Python3を使わないワークフロー

Python3を使わない、通常の記述は、右のようになります。
GUIでの記述は、求解エンジンへ送られて、解出力されます。

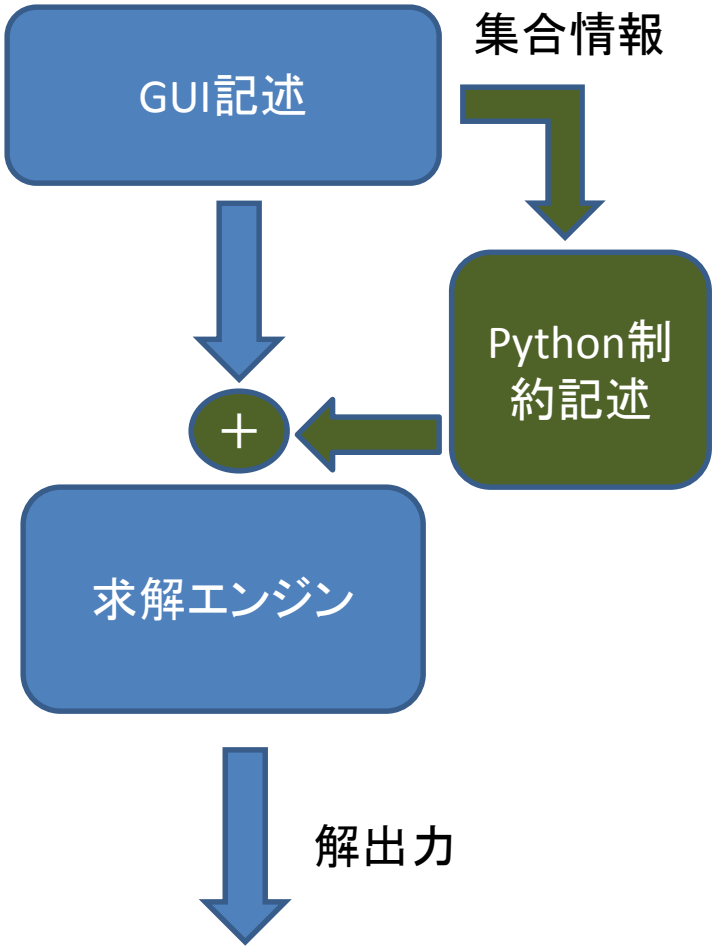


Python3を使ったワークフロー

Python3を使った記述は、GUIで記述した集合情報を入力として制約を記述します。

GUIで入力集合は記述済みですので、最小限の記述で制約を追加することが出来ます。

アドオンで、制約を追加できるということです。またチェックボックスで簡単に切り離すことも出来ます。



サポートするPython・スケナスVersion

32ビット・64ビット共　Python 3.68　です。

スケジュールナースバージョンは、１２７A以降です。

Python記述する場合の注意点

■スタッフプロパティに限らず、全てのプロジェクト記述の注意点なのですが、pythonで言語制約記述を行う場合、殆どすべての項目をそのままpython変数として利用しています。その場合、以下の事項に留意してください。　（python制約を使用する予定がなければ、無視して構いません）

■python制約で使えない記述

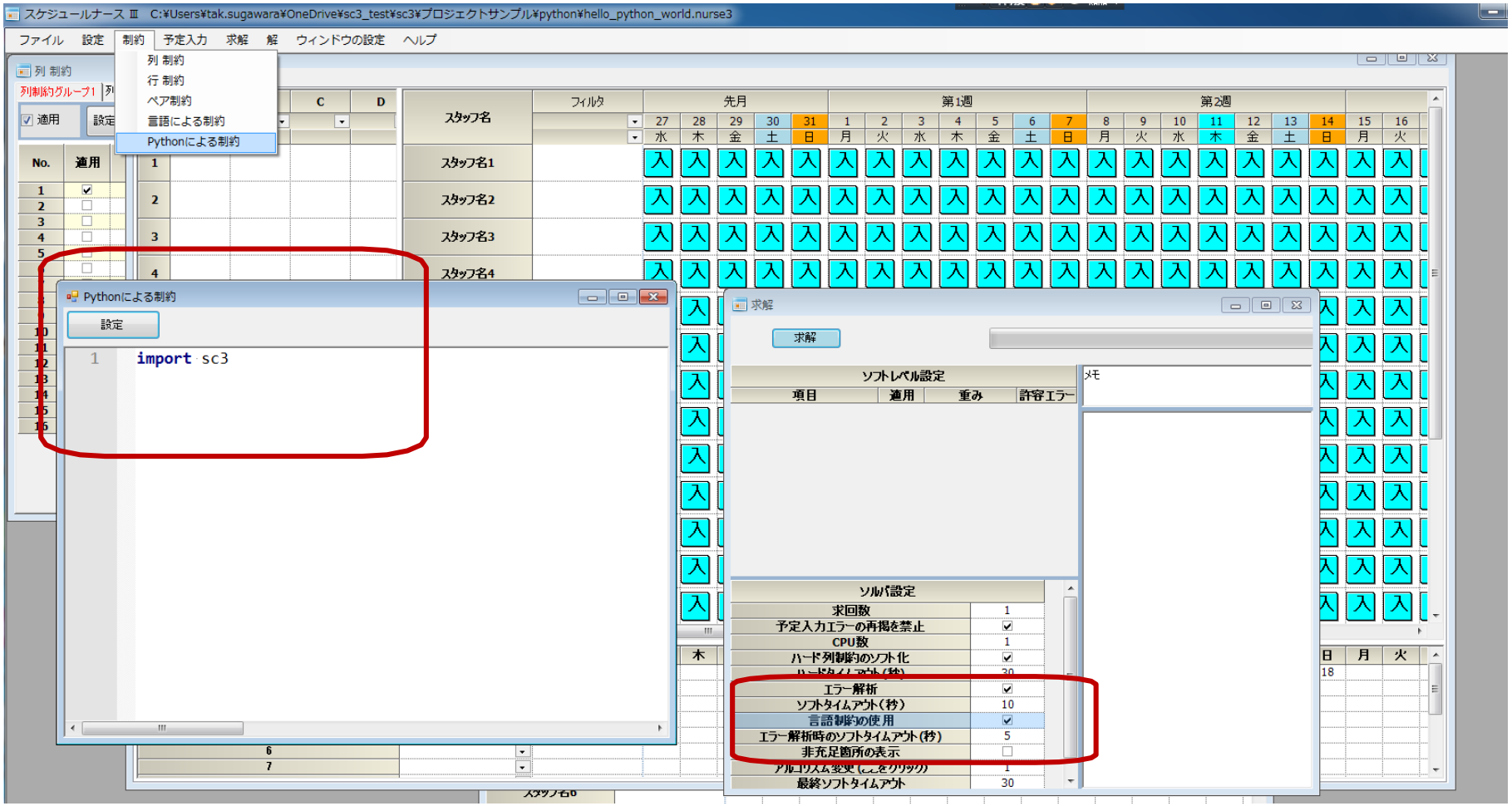
- 夜＊夜　　＊は使用不可
- 夜＊＊夜
- (特定増減日外)　（は使用不可
- [日]　[]は使用不可
- 10人日　数字が頭に来る記述は、半角全角問わず不可
- 日勤（拘束）　スペース不可　(不可
- 【標準】　【 不可　 】不可
- 4週n休前半　数字が頭は不可
- 新人準夜人数★　★は、不可
- 七-8人日　-不可
- チューター—数　－不可
- 160.25 .ドット　（小数点）不可
- 新人・班あり　・不可
- は、　、使用不可

使えないキャラクタは、スキップされて接頭辞が付加されます。これにより元の名前をなるべく残しつつコンパイルエラーを避けています。

■列制約については、適用チェックを外しても、Pythonプロパティファイルが生成されます。このため、適用しない制約でも、python上でイリーガルが記述があるとコンパイルエラーが発生します。

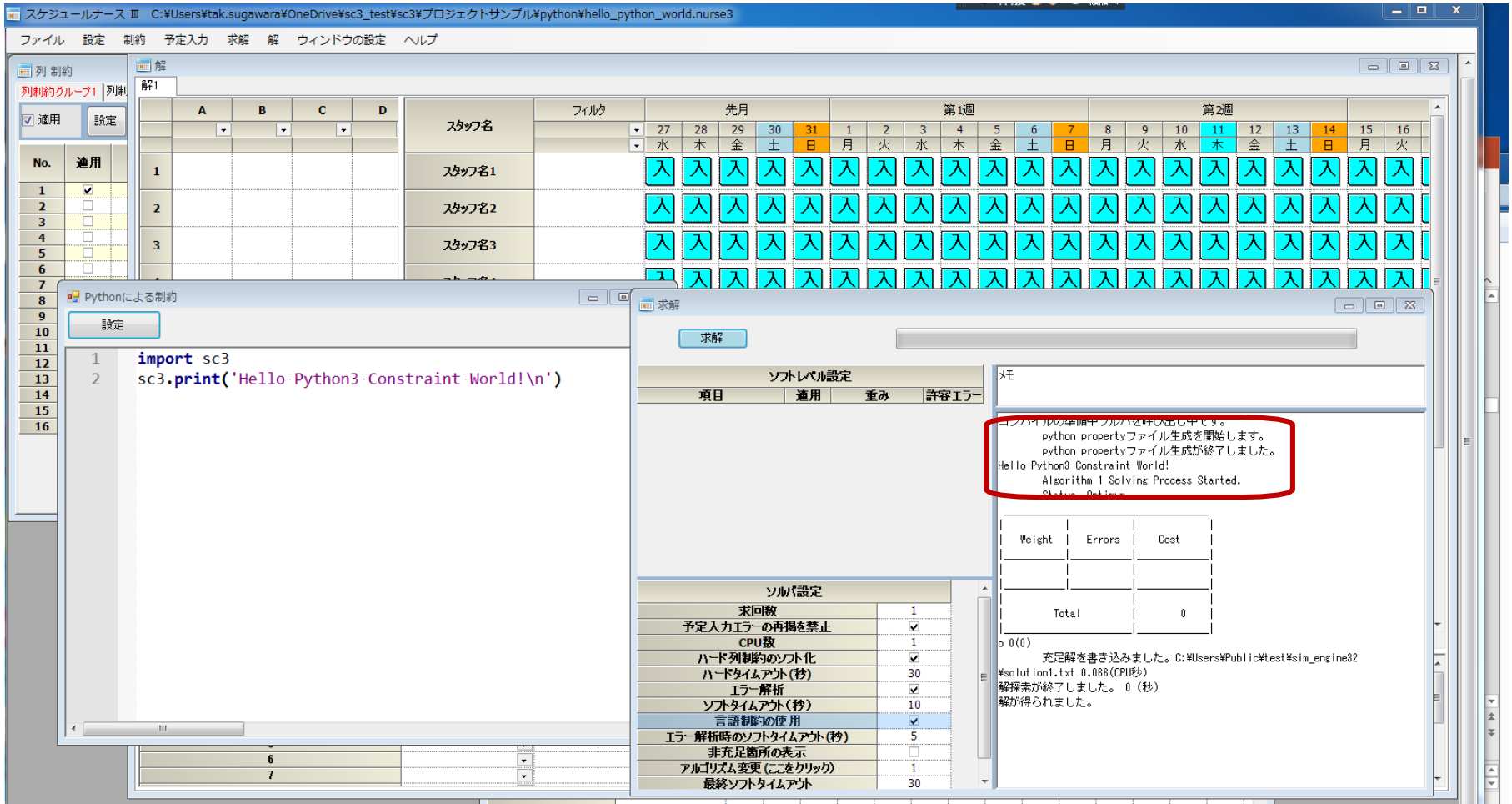
Python制約のEnable

■　言語制約の使用をチェックします。
■　Pythonによる制約でPython　Editorを起動し、import sc3 と打ち込んで設定をクリックします。



Hello World!

プロジェクトサンプルフォルダ/pythonフォルダにあるhello_python_worldプロジェクト(nurse3拡張子)をロードして求解すると右ペインにHello..が出力されます。



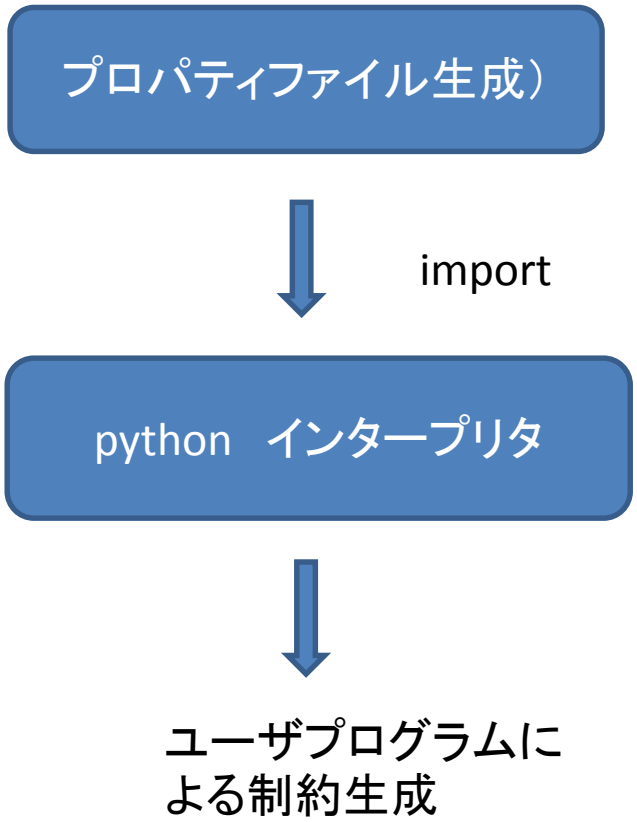
プロパティファイルの生成

python言語がEnableされて求解ボタンが押されると、プロパティファイルが生成されます。

ユーザが作成したPythonプログラムの先頭には、

import project_name_property.py

が暗黙に打ち込まれていて、プロパティファイルを読み込んだ状態でユーザプログラムが始まります。



プロパティファイルの内容

GUIで記述した内容の参照が出来ます。このソースファイルは、プロジェクトと同じフォルダに出力されます。このファイルを参照しながら、pythonで、制約を記述することになります。

```
#staffdef
staffdef=['スタッフ名1','スタッフ名2','スタッフ名3','スタッフ名4','スタッフ名5','スタッフ名6','スタッフ名7','スタッフ名8','スタッフ名9','スタッフ名10','スタッフ名11','スタッフ名12','スタッフ名13','スタッフ名14','スタッフ名15','スタッフ名16','スタッフ名17','スタッフ名18']
#daydef
制約開始日=5
制約終了日=35
表示開始日=0
daydef=['2016-07-27','2016-07-28','2016-07-29','2016-07-30','2016-07-31','2016-08-01','2016-08-02','2016-08-03','2016-08-04','2016-08-05','2016-08-06','2016-08-07','2016-08-08','2016-08-09','2016-08-10','2016-08-11','2016-08-12','2016-08-13','2016-08-14','2016-08-15','2016-08-16','2016-08-17','2016-08-18','2016-08-19','2016-08-20','2016-08-21','2016-08-22','2016-08-23','2016-08-24','2016-08-25','2016-08-26','2016-08-27','2016-08-28','2016-08-29','2016-08-30','2016-08-31']
#staffcollection
全スタッフ=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
入り=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
公休=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
日勤=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
明け=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
タスク1=[]
タスク2=[]
タスク3=[]
NoTaskVar=[]

#daycollection
今月=[5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35]
日=[4,11,18,25,32]
月=[5,12,19,26,33]
```

Python チュートリアル１

ｓｃ３は、ｐｙｔｈｏｎインタープリタにスケジュールナース専用のＡＰＩを組み込むためのライブラリです。全ての専用命令は、ｓｃ３が必要です。そのために、ユーザプログラム冒頭に、ｉｍｐｏｒｔ ｓｃ３が必要となります。
Ｐｙｔｈｏｎチュートリアル１をロードしてください。

Pythonによる制約

設定

```
1  import sc3
2  sc3.print('Hello Python3 Constraint World!\n')
3
4  for day in 全日:
5      sc3.print( 'day index'+str(day)+' は '+daydef[day]+' です。\n')
6
```

python変数　全日　と1次元配列daydefは、プロパティファイル中にあります。
print文では、文字列に変換(str())して、表示してください。

Pythonチュートリアル１ログ

次のようなログが生成されます。pythonインタープリタが終了してから、求解エンジン本体が起動されます。

```
        python propertyファイル生成を開始します。
        python propertyファイル生成が終了しました。
Hello Python3 Constraint World!
day index0 は 2016-07-27 です。
day index1 は 2016-07-28 です。
day index2 は 2016-07-29 です。
day index3 は 2016-07-30 です。
day index4 は 2016-07-31 です。
..
Algorithm 1 Solving Process Started.
        Status  Optimum.
```

エラー　行noは、1行前

1行から数行書く毎に大体エラーが出るものです。下は、エラーの例です。

SyntaxError: ('invalid character in identifier', ('<string>', 6, 56, "　scs.print('day index'+str(day)+'は’+daydef[day]+'です。¥¥n')¥n"))解探索が終了しました。12 (秒)
解を見つけることが出来ませんでした。

Python処理系のエラーをそのまま出しています。上の例で、6,56では、行No、列Noに対応しています。

暗黙にインポートしているライブラリの関係で、エラー行は、表示より1行ずれています。補正してください。表示で6とあったとき、正しいソース行は、5です。

Pythonが初めての方は、インデント系のエラーが多いと思います。
また、次のようにu3000は、全角スペースがあるエラーとなります。

SyntaxError: ('invalid character in identifier', ('<string>', 8, 21, '
sc3.print(s)¥u3000#Debug¥n'))解

GetShiftVar　Python チュートリアル２

■スケジュールナースの各シフトは、OneHot　エンコーディングされています。各人、各日にシフト数だけのバイナリ変数があります。つまり、１になるシフトは、各人、各日あたり1個以上、1個以下です。

■ユーザは、任意のシフト変数を次の書式で、取り出すことが出来ます。

```
v=sc3.GetShiftVar(person,day,shift)
```

ここで、person/dayは、インデックス変数

シフトは、インデックス変数または、シフトストリングです。

AddHard　Python チュートリアル２

■ハード制約を追加する関数です。次の書式です。

sc3.AddHard(v,string)

vは、GetShiftVarで取得したシフトVariable、またはsc3サポート制約関数または、And/Or/Not/SeqExpr/SeqCompの出力変数になります。

Pythonによる制約

設定

```
import sc3

for person in 全スタッフ:
    for day in 月:
        v=sc3.GetShiftVar(person,day,'公休')
        s='言語公休設定_'+staffdef[person]+' '+daydef[day]+' \n' #制約Keyword +Space + Details
        sc3.print(s) #Debug
        sc3.AddHard(v,s) #AddHard(v,'')でも可だが、解が無い場合の原因解析を容易にするためString情報を付加
```

vは、各スタッフ月曜日の公休という変数になります。AddHard、これをassertする制約になります。

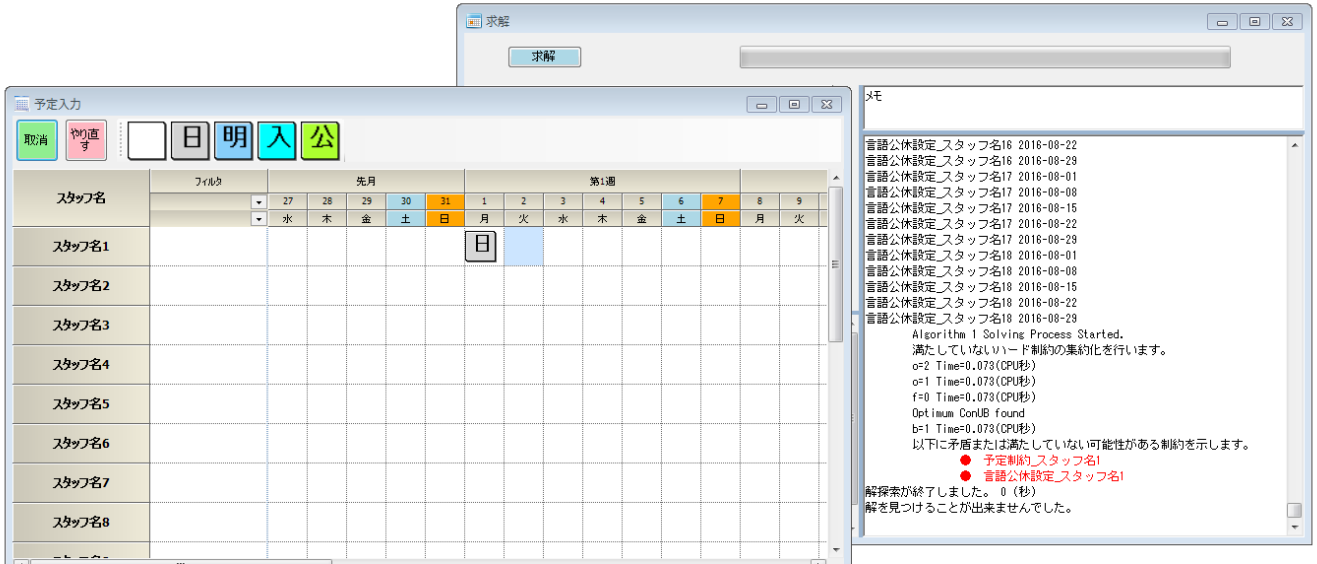
AddHard　Python チュートリアル２　ログ

確かに、全スタッフの月曜日の勤務を公休とすることが出来ました。

スタッフ名	フィルタ	先月					第1週							第2週								
		27	28	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
		水	木	金	土	日	月	火	水	木	金	土	日	月	火	水	木	金	土	日	月	火
スタッフ名1		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名2		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名3		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名4		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名5		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名6		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名7		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名8		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名9		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名10		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名11		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名12		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入
スタッフ名13		入	入	入	入	入	公	入	入	入	入	入	入	公	入	入	入	入	入	入	公	入

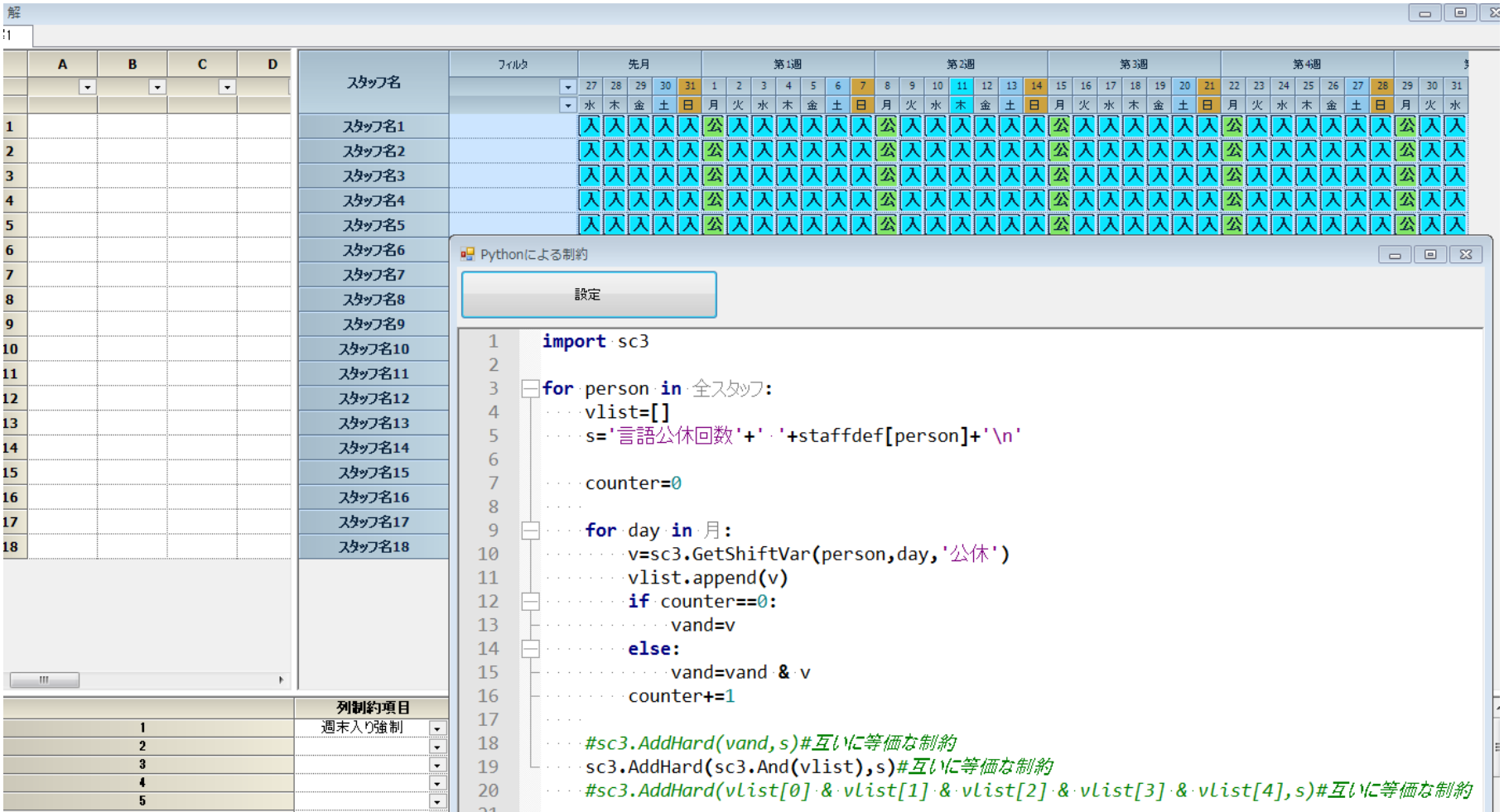
AddHard　Python チュートリアル２　故意に解なしにする

月曜日は、言語制約で全部公休としました。それに対して、日勤勤務を月曜日に入れてみました。言語制約と予定制約は矛盾しているので、当然解はありません。
右ペインを見て頂きたいのですが、言語公休制約_スタッフ1というところまで、原因を割り出しています。これは、AddHardのところのstringが功を奏した例です。解がないときの原因として列挙されないと、悩むことが多いので設定することを強くお勧めします。Stringの書式は、
1次元の場合　Keyword＋SPACE+variable_words
2次元の場合　Keyword+variable_words+SPACE+variable_words



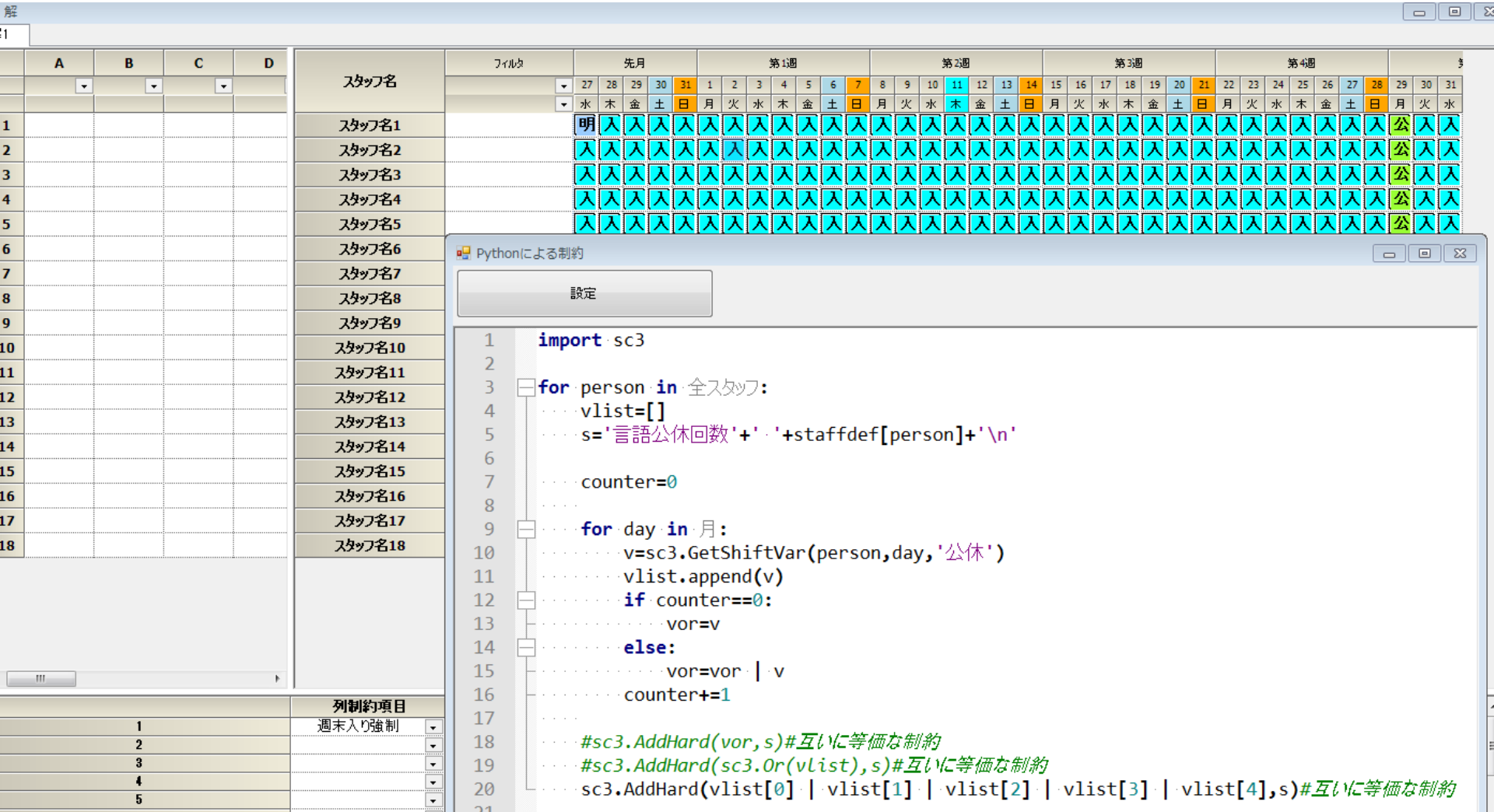
AddHard　Python チュートリアル３　And制約

下3行は、どれもValidで等価な制約です。全ての月曜日が公休となる制約を記述しています。



AddHard　Python チュートリアル４　Or制約

下3行は、どれもValidで等価な制約です。少なくとも1個の月曜日が公休となる制約を記述しています。



5個の月曜日の公休のうち、4個だけ公休を取るという制約を考えてみましょう。これを論理演算子だけで制約せよ！　という例題をやってみましょう。

次のA）B)を実現することで、実装可能です。
■A)Σ~公休＞ = １ですから、~公休 のORをとってAssertすればよいです。

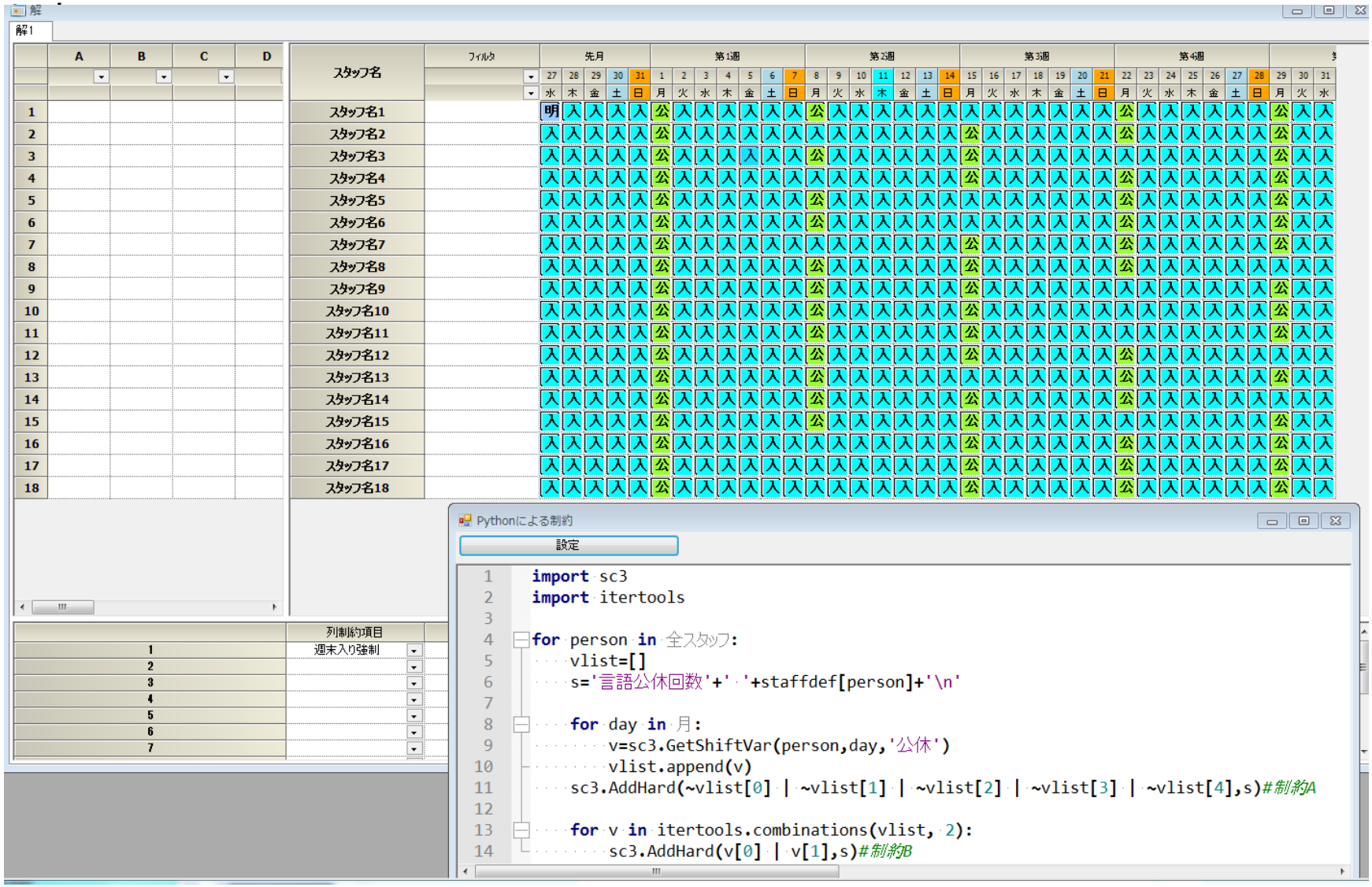
■B)5個から2個取る全ての組み合わせで2個とも~公休は禁止　→~(~公休 & ~公休)　→公休　|　公休　となりますから、全ての組み合わせでORをAssertすればよいです。

5個から２個取る全ての組合せを列挙するのが面倒ですが、それは、Python Itertoolsに任せればよいです。Itertoolsが全ての組み合わせを列挙してくれます。

面倒な仕様にも思えますが、次のページのソースコードのように僅か１４行で記述できます。

AddHard　Python チュートリアル５　And ・Or・Not制約

５個の月曜日のうち１個だけ公休ではない日がある　という仕様を実現できま



AddHard　Python チュートリアル６　基数制約

不等式の制約を基数制約といいます。チュートリアル４では、Xi=公休

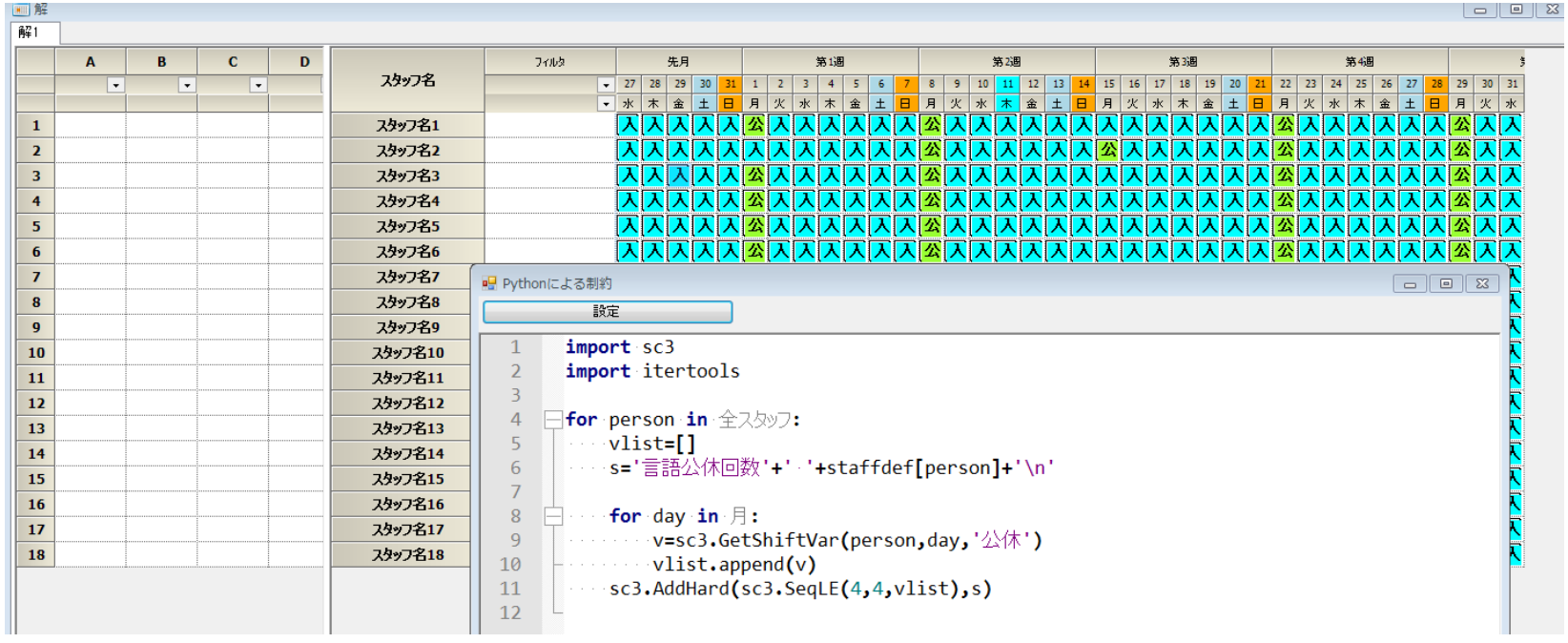
　　Σ~Xi<=4　AND Σ~Xi>=4

をAnd,Or,Notの論理式だけを使って実現しました。が、基数制約には、専用の関数が用意されています。

　　基数制約　ΣXi　>=min & Σ　Xi<=maxは、

　　sc3.SeqLE(min,max,List)

Listは、シフトVariable、または、他のAnd/Or,Not等の演算結果によるVariableである必要があります。



世界一難しい数独　その１

基数制約で実装されています。なお、解を二つ求めようとしますが、唯一の解しか存在しないため、solution2.txtは、存在しない..メッセージが出ます。

スタッフ名	フィルタ	第1週			
		1	2	3	4
		金	土	日	月
スタッフ名1		1	4	5	3
スタッフ名2		8	3	9	6
スタッフ名3		6	7	2	9
スタッフ名4		4	9	6	1
スタッフ名5		2	1	8	4
スタッフ名6		7	5	3	2
スタッフ名7		3	6	7	5
スタッフ名8		9	8	4	7
スタッフ名9		5	2	1	8

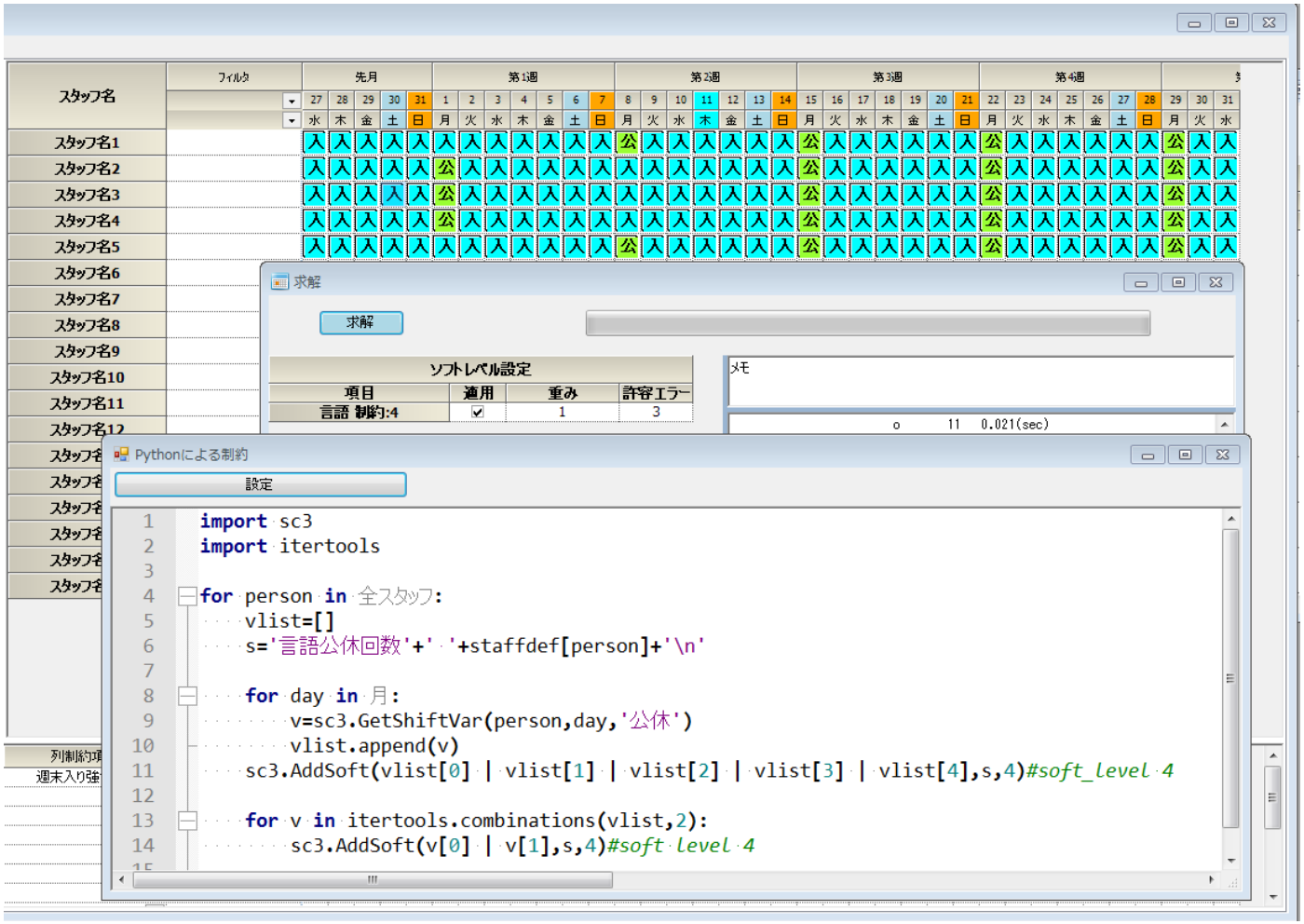
Pythonによる制約

設定

```
import sc3
for day in 全日: #列制約
    for シフト in 全シフト:
        V=[]
        for 人 in 全スタッフ:
            V.append(sc3.GetShiftVar(人,day,シフト))
        sc3.AddHard(sc3.SeqLE(1,1,V),'')
for 人 in 全スタッフ: #行制約
    for シフト in 全シフト:
        V=[]
        for day in 全日:
            V.append(sc3.GetShiftVar(人,day,シフト))
        sc3.AddHard(sc3.SeqLE(1,1,V),'')
for 人 in スタッフブロックトップ: #ブロック制約
    for シフト in 全シフト:
        for day in ブロックトップ日集合:
            V=[]
            for i in range(3):
                for j in range(3):
                    V.append(sc3.GetShiftVar(人+i,day+j,シフト))
            sc3.AddHard(sc3.SeqLE(1,1,V),'')
```

sc3.AddSoft(variable ,string ,soft_level)が書式です。Soft_levelは、定数である必要があります。最初設定すると、求解左ペインのソフトレベルのチェックボックスが空欄なので、チェックして求解します。

。



SeqError　Python チュートリアル８

SeqError(min,max, allowable_errors,list)が書式です。常にAddSoftと組で使います。下の結果を見ると公休４個に設定しているはずですが、そうなっていないのは、日勤が予定入力として入力されているからです。このエラーを何個まで許すかが、allowable_errorsです。

Pythonによる制約

設定

```
import sc3

for person in 全スタッフ:
    vlist=[]
    s='言語公休回数'+' '+staffdef[person]+'\n'
    for day in 月:
        v=sc3.GetShiftVar(person,day,'公休')
        vlist.append(v)
    sc3.AddSoft(sc3.SeqError(4,4,4,vlist),s,4) #min max allowable errors list
```

SeqError　Python チュートリアル８

allowable_errorsを試しに３と設定してみると、次のように解がない状態になってしまいます。許容エラー数を超えた日勤の入力箇所があるためです。min-allowable_errors, max+allowable_errosの境界上にハード制約が設定されます。これを用いて、際限なくソフト化することの防止、強制的な平準化を行うことができます。しかし、同時に解がないというリスクを負うことも意味します。

設定

```
2
3  for person in 全スタッフ:
4      vlist=[]
5      s='言語公休回数'+' '+staffdef[person]+'\n'
6      for day in 月:
7          v=sc3.GetShiftVar(person,day,'公休')
8          vlist.append(v)
9      sc3.AddSoft(sc3.SeqError(4,4,3,vlist),s,4) #min max allowable errors list
```

求解

ソフトレベル設定

項目	適用	重み	許容エラー
言語 制約:4	☑	1	3

メモ

コンパイルの準備中ソルバを呼び出し中です。
python propertyファイル生成を開始します。
python propertyファイル生成が終了しました。
Algorithm 1 Solving Process Started.
解がありません。ソフト制約の許容エラー数を大きくするとエラーが解消される可能性が高いです。
解探索が終了しました。 0 (秒)
解を見つけることが出来ませんでした。

予定入力

取消

スタッフ名

スタッフ名1
スタッフ名2
スタッフ名3
スタッフ名4
スタッフ名5
スタッフ名6

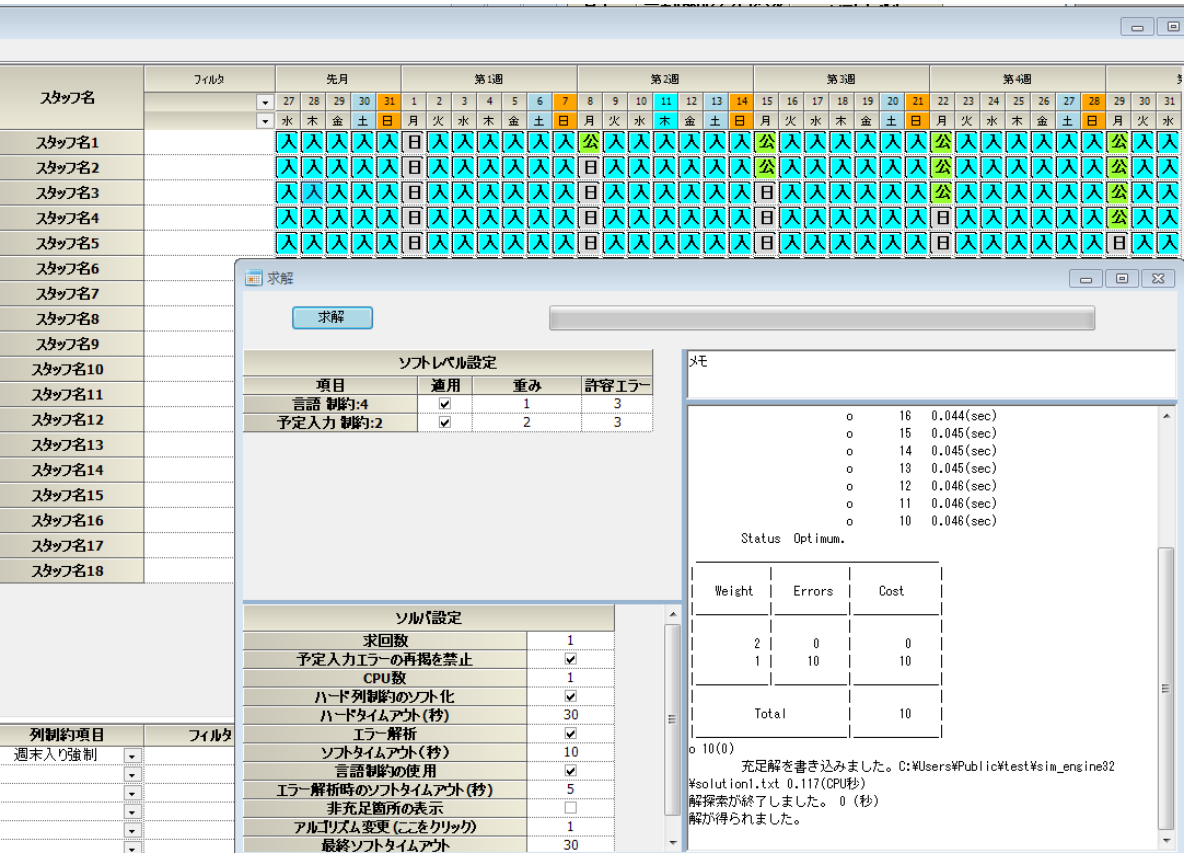
ソルバ設定

求回数	1
予定入力エラーの再掲を禁止	☑
CPU数	1
ハード列制約のソフト化	☑
ハードタイムアウト(秒)	30
エラー解析	☑
ソフトタイムアウト(秒)	10

SeqError　Python チュートリアル９

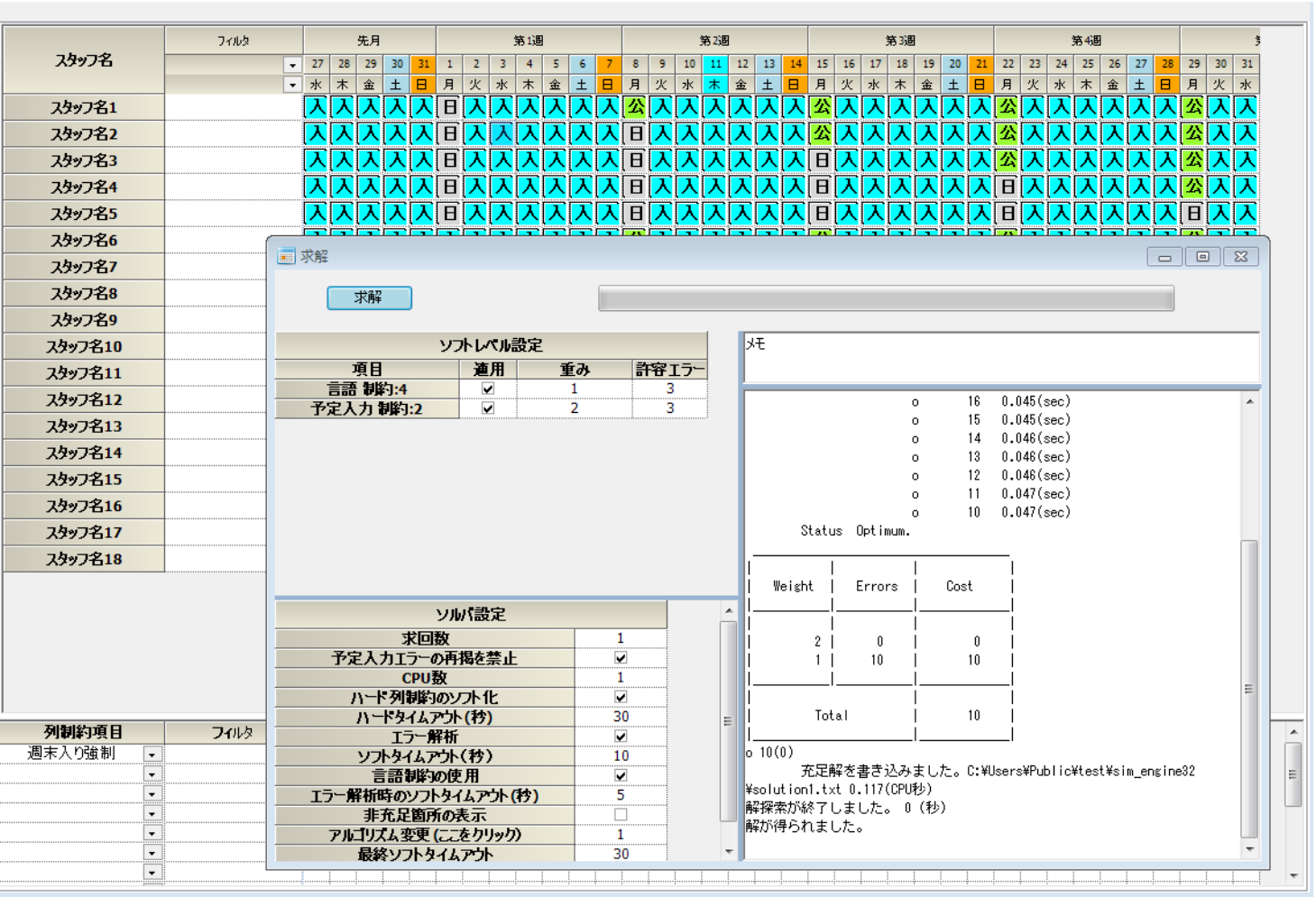
■ GUIで設定した基数制約には、求解左ペインの許容エラーで、allowable_errosの値を変えることが出来ますが、言語制約については、適用されません。

■ソルバは、Totalの目的関数値をできるだけ小さくしようと割り当てます。その結果全てを満たさないまでも、出来る限る理想値に近づくように配置されます。その結果、下のようになることは、必然となります。



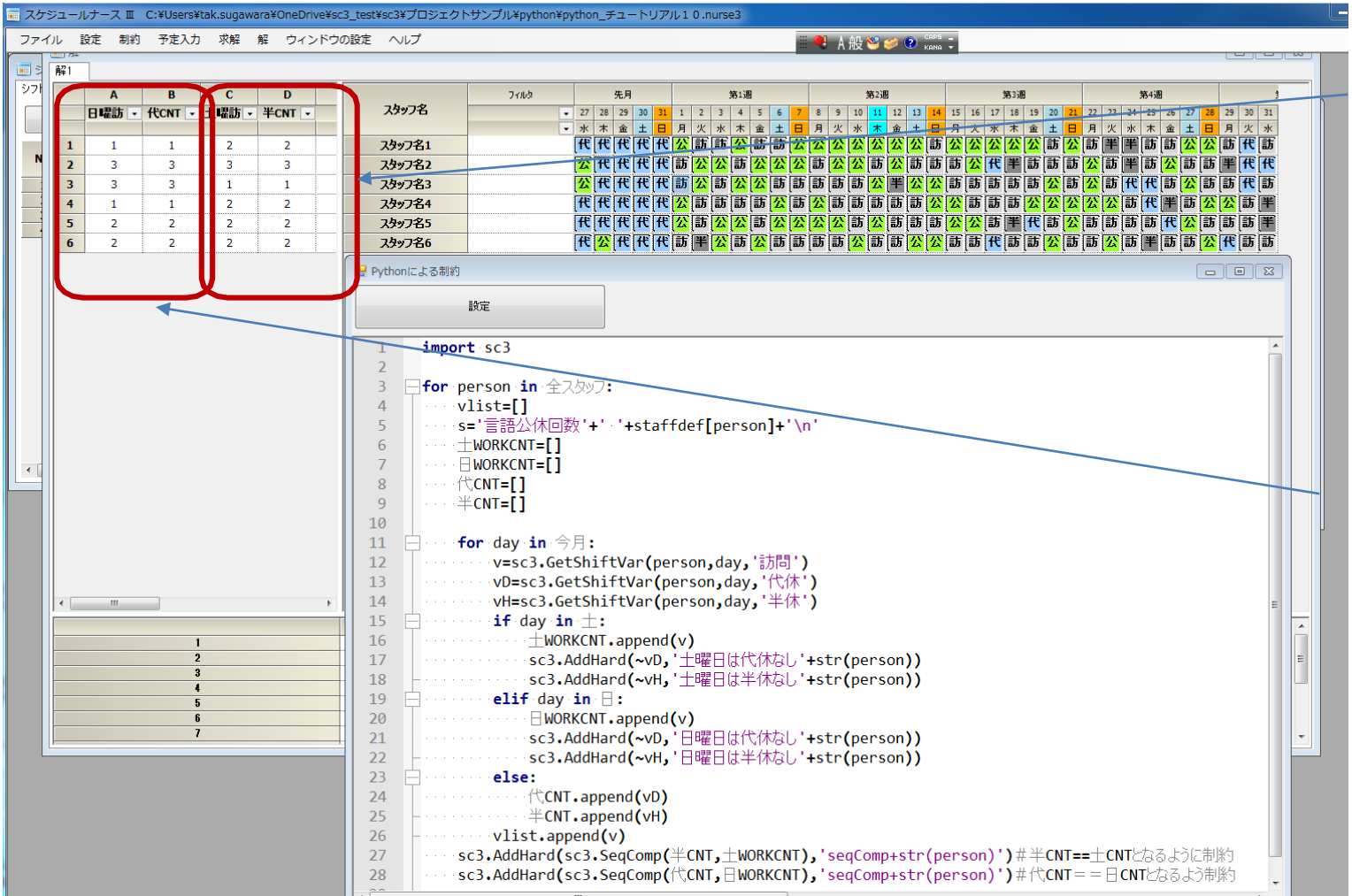
SeqError　Python チュートリアル９重み変更

■ 予定制約と基数制約の重みを交換すると、基数制約の重みが効いて、予定が変更され、基数制約については、エラーなしとなります。このようにPythonソースをいじることなく重み変更も可能です。



SeqComp　Python チュートリアル１０

■ SeqComp(List1,List2)　List1の要素をX(i)、List2の要素をY(i)とするとき
ΣX(i)==ΣY(i)のとき、Trueを返す関数です。AddHardと組み合わせるとΣX(i)==ΣY(i)
となるように制約します。



土曜日の訪問数は、その月の半休数に等しい。

日曜日の訪問数は、その月の代休数に等しい

という制約になります。

Python チュートリアル１１　予定シフトとスタッフプロパティ

■ Excelフォルダにあるサンプルです。スタッフプロパティで、数字だけで構成されたプロパティ(下図で公休数)は、Python　Dictionaryとして取り出すことが出来ます。これを利用してスタッフ毎の各種設定を行うことが出来ます。

スタッフ定義 | スタッフ毎のシフト | グループ定義 | グループ集合 | スタッフ毎のタスク

設定

No.	適用	スタッフ名	公休数
1	☑	A1	
2	☑	A2	13
3	☑	A3	13
4	☑	A4	13
5	☑	A5	13
6	☑	A6	13
7	☑	A7	13
8	☑	A8	13
9	☑	A9	13
10	☑	A10	13
11	☑	A11	15
12	☑	A12	13
13	☑	A13	13
14	☑	A14	13
15	☑	A15	13
16	☑	A16	13
17	☑	A17	13
18	☑	A18	13
19	☑	A19	13
20	☑	A20	13
21	☑	A21	13
22	☑	A22	13
23	☑	A23	17
24	☑	A24	13
25	☑	A25	13
26	☑	A26	13
27	☑	A27	
28	☑	A28	
29	☑	A29	
30	☑	A30	
31	☑	A31	
32	☑	A32	

金	土	日	月	火	水	木
師	半	休	夜	明	休	休
日	日	休	休	休	休	休

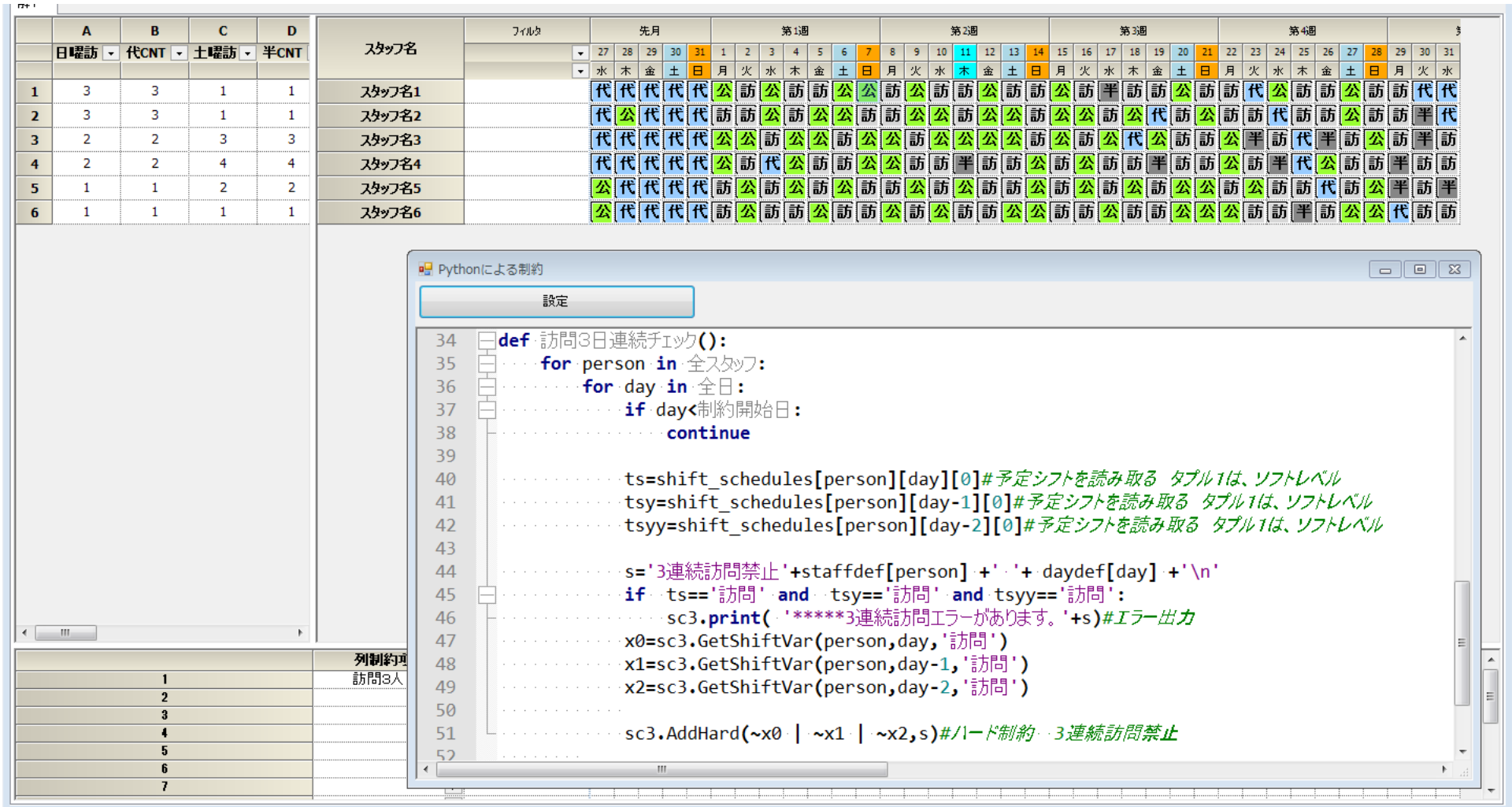
Pythonによる制約

設定

```
1  import sc3
2  def 公休数設定():
3  ⟶for person in 公休数.keys():
4  ⟶⟶value=公休数[person]
5  ⟶⟶sc3.print(staffdef[person]+'の公休数は'+str(value)+'回に制約\n')
6  ⟶⟶VList=[]
7  ⟶⟶for day in 全日:
8  ⟶⟶⟶if day<制約開始日:
9  ⟶⟶⟶⟶continue
10 ⟶⟶⟶v1=sc3.GetShiftVar(person,day,'公休')
11 ⟶⟶⟶VList.append(v1)
12 ⟶⟶s="person"+str(person)+"公休数"
13 ⟶⟶sc3.AddSoft(sc3.SeqError(value,value,7,VList),s,7) #許容エラー7 いい加減に設定される可能性がある
14
15 def 週あたりの勤務回数設定():
16 ⟶for person in 週あたりの勤務回数.keys():
17 ⟶⟶n=週あたりの勤務回数[person]
18 ⟶⟶sc3.print(staffdef[person]+'の週当たりの勤務回数は'+str(n)+'回に制約\n')
19 ⟶⟶for mon in 月:#月曜日を起点に1週間見る
20 ⟶⟶⟶sc3.print('月曜日 '+str(mon)+'\n')
21 ⟶⟶⟶VList=[]
22 ⟶⟶⟶for d in range(7):
23 ⟶⟶⟶⟶day=mon+d
24 ⟶⟶⟶⟶if day >制約終了日:
25 ⟶⟶⟶⟶⟶break
26 ⟶⟶⟶⟶else:
27 ⟶⟶⟶⟶⟶v1=sc3.GetShiftVar(person,day,'休日集合')#
28 ⟶⟶⟶⟶⟶VList.append(~v1)#休日の否定は勤務
```

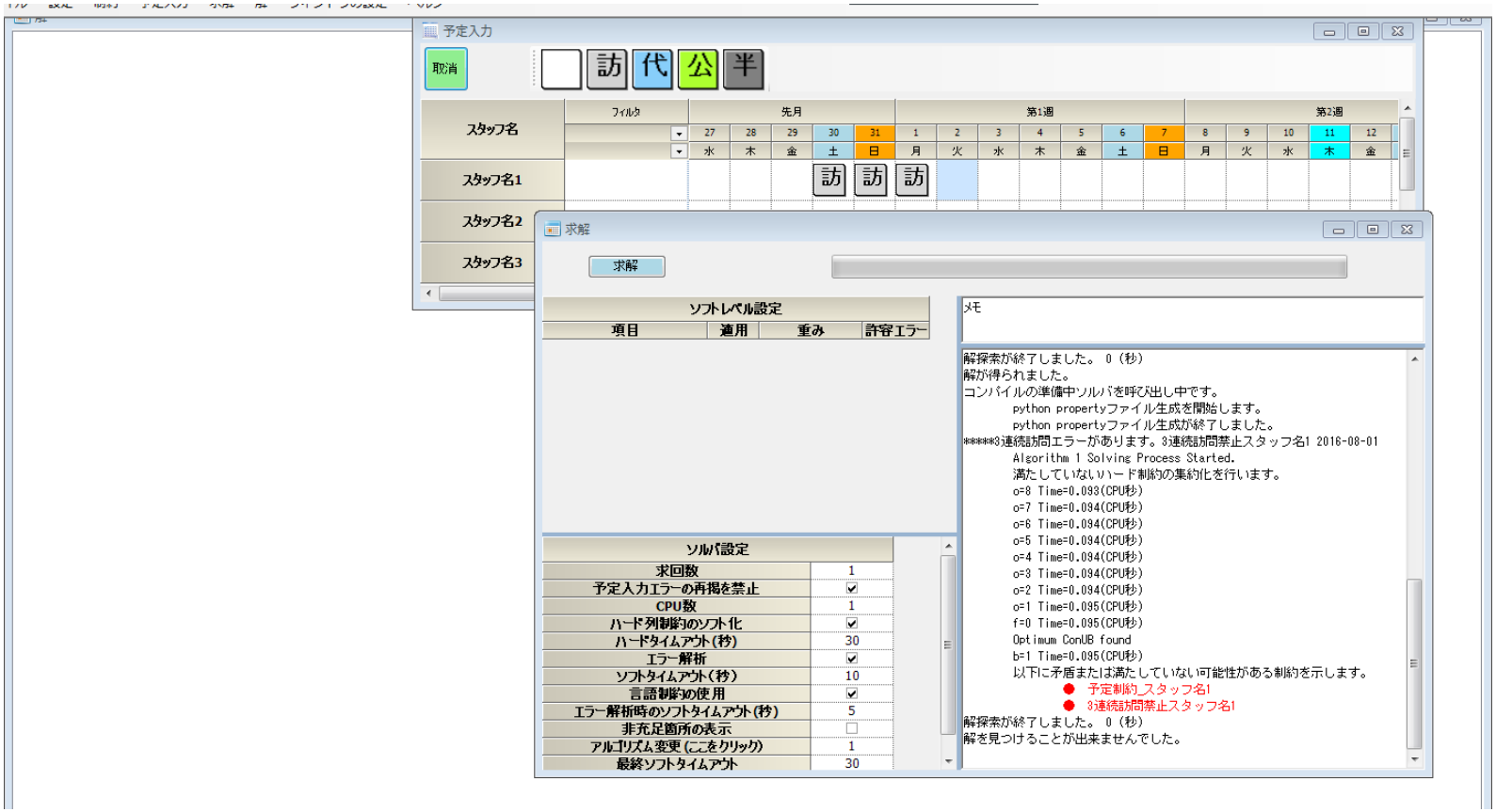
Python チュートリアル１２　予定シフト

■ 単に制約するだけではなく、予定シフトを読み込んでエラーを出力させます。



Python チュートリアル１２　予定シフトエラー

■ 予定シフトのエラーが検出を行い、制約エンジンに行く前にエラーを表示させることが出来ます。別な応用としては、予定シフトとスタッフプロパティを読み込んで、予定シフトが書き込まれていないときに限り、スタッフプロパティによるDefaultシフトを書き込むことが考えられます。ソフト制約を使えば静的には、実現可能ですが、動的に判断した方が、余計なソフトエラーはなくなりメンテナンス性が良くなる場合もあります。



Python チュートリアル１３　スタッフプロパティー基数制約

■ スタッフプロパティで、スタッフ毎の公休数を個別設定する例です。
下記は、Excelから読み込んだスタッフプロパティです。この例では、スタッフ毎の契約が異なるために、個別に公休数を設定する必要があります。

このように、数字だけで構成されたプロパティは、Propertyファイルで、辞書として取り出すことができます。
（数値以外が一項でもあると辞書は生成されません。ブランクは可）

下は、Propertyファイルの中身です。ゼロベースのperson_indx:公休数の項が、辞書の要素になります。Pythonプログラムをこれを拾ってきて制約すれば良い訳です。
公休数
={1:13,2:13,3:13,4:13,5:13,6:13,7:13,8:13,9:13,10:15,11:13,12:13,13:13,14:13,15:13,16:13,17:13,18:13,19:13,20:13,21:13,22:17,23:13,24:13,25:13}

No.	適用	スタッフ名	コメント	公休数	全スタッフ属性
1	☑	A1			全スタッフ
2	☑	A2		13	全スタッフ
3	☑	A3		13	全スタッフ
4	☑	A4		13	全スタッフ
5	☑	A5		13	全スタッフ
6	☑	A6		13	全スタッフ
7	☑	A7		13	全スタッフ
8	☑	A8		13	全スタッフ
9	☑	A9		13	全スタッフ
10	☑	A10		13	全スタッフ
11	☑	A11		15	全スタッフ
12	☑	A12		13	全スタッフ
13	☑	A13		13	全スタッフ
14	☑	A14		13	全スタッフ
15	☑	A15		13	全スタッフ
16	☑	A16		13	全スタッフ
17	☑	A17		13	全スタッフ
18	☑	A18		13	全スタッフ
19	☑	A19		13	全スタッフ
20	☑	A20		13	全スタッフ
21	☑	A21		13	全スタッフ
22	☑	A22		13	全スタッフ
23	☑	A23		17	全スタッフ
24	☑	A24		13	全スタッフ
25	☑	A25		13	全スタッフ
26	☑	A26		13	全スタッフ
27	☑	A27			全スタッフ
28	☑	A28			全スタッフ
29	☑	A29			全スタッフ
30	☑	A30			全スタッフ
31	☑	A31			全スタッフ

Python チュートリアル１３　スタッフプロパティー基数制約ソース

■ SeqError(Soft)または、SeqLE(Hard)のみ、ＧＵＩに計数表示させることが出来ます。
■ 計数表示するには、AddSoftまたは、AddHardに渡すstring が次の書式である必要があります。ＧＵＩ表示はAlgorithm1のみのサポートです。
行表示の場合
■ Keyword +space +staffdef[index]　または、Keyword+space+’person’+index
列表示の場合
■ Keyword +space +daydef[index]　または、Keyword+space+’day’+index

```
import sc3
def 公休数設定():
    for person in 公休数.keys():
        value=公休数[person]
        VList=[]

        for day in 全日:
            if day<制約開始日:
                continue
            v1=sc3.GetShiftVar(person,day,'公休')
            VList.append(v1)

        if person in 長期休み:#長期休みは除外　長期休みはスタッフプロパティで設定
            continue
        sc3.print(staffdef[person]+'の公休数は'+str(value)+'回に制約\n')
        s="公休数Python"+' '+staffdef[person]#Keyword +space +variable words
        sc3.AddSoft(sc3.SeqError(value,value,7,VList),s,7) #許容エラー7　いい加減に設定される可能性がな
        #sc3.AddHard(sc3.SeqLE(value,value,VList),s)
```

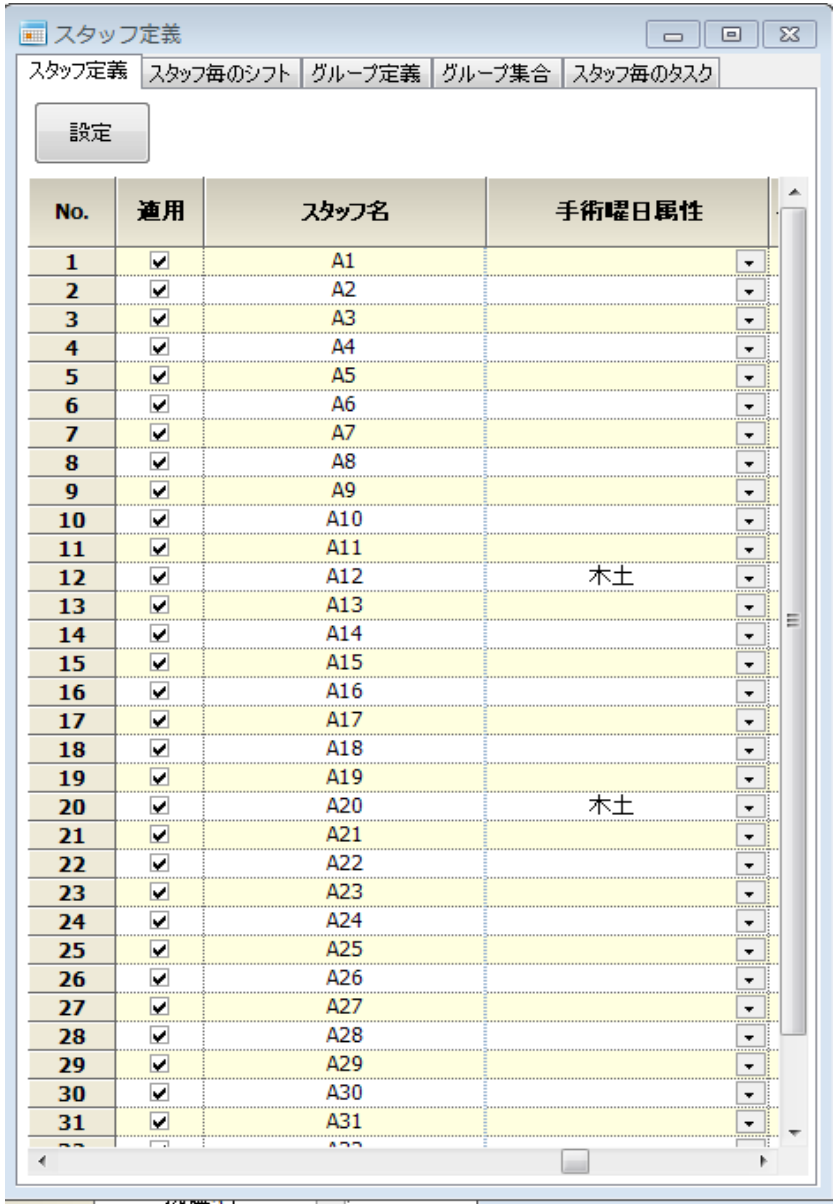
Python チュートリアル１３　スタッフプロパティー曜日設定

■ スタッフプロパティで、曜日設定する例です。左は、Excelから読み込んだスタッフプロパティです。手術可能な曜日が木土に設定されるスタッフがいます。これが将来、木土以外に追加変更されるときのGUIメンテナンスの手間を想像しましょう。面倒ですね。

項全部が曜日であるときに、辞書として取り出すことが出来ます。（**曜日以外が一項でもあると辞書は生成されません。ブランクは可**）

下は、Propertyファイルの中身です。ゼロベースのperson_indx:曜日の項が、辞書の要素になります。Python　プログラムをこれを拾ってきて制約すれば良い訳です。
木土=[1,6,8,13,15,20,22,27,29,34,36]

手術曜日属性={11:木土,19:木土}



No.	適用	スタッフ名	手術曜日属性
1	☑	A1	
2	☑	A2	
3	☑	A3	
4	☑	A4	
5	☑	A5	
6	☑	A6	
7	☑	A7	
8	☑	A8	
9	☑	A9	
10	☑	A10	
11	☑	A11	
12	☑	A12	木土
13	☑	A13	
14	☑	A14	
15	☑	A15	
16	☑	A16	
17	☑	A17	
18	☑	A18	
19	☑	A19	
20	☑	A20	木土
21	☑	A21	
22	☑	A22	
23	☑	A23	
24	☑	A24	
25	☑	A25	
26	☑	A26	
27	☑	A27	
28	☑	A28	
29	☑	A29	
30	☑	A30	
31	☑	A31	

Python チュートリアル１３　予定が入っていたら制約しない

■ ソースプログラムは、当該スタッフの木土以外を手術禁止にします。その際、予定も読み込んで、予定が入っていれば、制約しないようにしています。例外はつきものですから。

■将来、木土以外の定義が追加されたとしても、その曜日定義さえ追加すればよいです。(運がよければ、定義済みでなにもしなくてもよい。定義がなければエラーとなり実装不備を認識できます）pythonプログラム本体の変更は必要ありません。

Pythonによる制約

設定

```
112 def 手術曜日設定():
113     for person in 手術曜日属性.keys():
114         for day in 全日:
115             if day<制約開始日:#制約開始日からしか見ない
116                 continue
117             if day not in 手術曜日属性[person]:
118                 ts=shift_schedules[person][day][0]
119                 if ts !='':#既に予定が入っていたら制約しない。予定優先。
120                     continue
121                 #予定が入っていないときだけ制約
122                 s='手術曜日設定_'+staffdef[person]+' '+str(day)
123                 v=sc3.GetShiftVar(person,day,'手術')
124                 sc3.AddHard(~v,s)#手術曜日以外はDeassert
```

Python チュートリアル１３　複雑な仕様の例

■ 仕様

手術シフト平日(日祝外）に２人必要。オペ看２人、オペ看と組んでできるプロパティグループ、オペ看と組まないでも可能な中堅プロパティグループがある。さらに、曜日指定、中堅プロパティグループ内での優先度が存在する

	手術看護師とペアグループ	手術看護師不在時グループ	優先度
オペ看2人	0	0	高い
オペ看1人	1	0	中
オペ看0人	0	2	低い

スタッフ定義

スタッフ定義 | スタッフ毎のシフト | グループ定義 | グループ集合 | スタッフ毎のタスク

設定

No.	適用	スタッフ名	手術看護師とペア可グループ	手術曜日属性	手術看護師不在時グループ	手術看護師不在時優先属性	病棟外来属性
1	☑	A1					病棟
2	☑	A2					病棟
3	☑	A3					病棟
4	☑	A4					病棟
5	☑	A5	手術可		○		病棟
6	☑	A6	手術可		○		病棟
7	☑	A7					病棟
8	☑	A8					手術
9	☑	A9					手術
10	☑	A10	手術可		○		病棟
11	☑	A11					病棟
12	☑	A12	手術可	木土			病棟
13	☑	A13	手術可		○		病棟
14	☑	A14	手術可		○		病棟
15	☑	A15	手術可		○		病棟
16	☑	A16	手術可		○		病棟
17	☑	A17	手術可		○	手術看護師不在時優先	病棟
18	☑	A18	手術可		○	手術看護師不在時優先	病棟
19	☑	A19			○		病棟
20	☑	A20	手術可	木土			病棟
21	☑	A21					病棟

Python チュートリアル１３　複雑な仕様の例　オペ看2人

■ 優先度をオペ看人数に対応させる

下表のように、オペ看人数2人をターゲットとする制約を行えばよいです。ソフト制約SeqError(min,max,allowable_errors,string) を使えば優先度の実装が出来ます。

	ペナルティ	優先度
オペ看2人	0	高い
オペ看1人	1	中
オペ看0人	2	低い

実装コードでは、予定で、オペ看の人数が既に確定しているときは、無用なソフトエラーをカウントさせないようにしています。

```
70  def オペ看2人(day):
71      ope_nurse_list=オペ看手術シフトリスト取得(day)
72      fixed_count=0
73      for person in 手術看護師:
74          ts=shift_schedules[person][day][0]
75          if ts !='手術':
76              fixed_count+=1
77      s="day"+str(day)
78      if fixed_count==0:#手術外が確定していなければ
79          sc3.AddSoft(sc3.SeqError(2,2,2,ope_nurse_list),'オペ看2人'+' '+s,5)#soft count対象
80      elif fixed_count==1:#手術外1人確定しているので、意味があるソフト重み最大は1
81          sc3.AddSoft(sc3.SeqError(1,1,1,ope_nurse_list),'オペ看1人'+' '+s,5)#soft count対象
```

Python チュートリアル１３　複雑な仕様の例　SeqExpr

■ オペ看が1人(A)ならば、オペ看ペアグループは1人(B)　です。AならばBは、~A|Bになります。A,B共ブール変数です。　基数制約min-maxの範囲内にあるときに、ブール変数を返すのが、SeqExprです。

SeqExpr(min,max,Type,List)

になります。Typeは、動作を指定します。

Σlist>=min　のとき、Type=1

Σlist<=max　のとき、Type=0

Σlist>=min && Σlist<=maxのとき、Type=2

```
83  def オペ看1人オペ看ペアグループより1人(day):
84      ope_nurse_list=オペ看手術シフトリスト取得(day)
85      A=sc3.SeqExpr(1,1,2,ope_nurse_list)
86
87      pairs_list=オペ看ペアグループの手術シフトリスト取得(day)
88      B=sc3.SeqExpr(1,1,2,pairs_list)
89
90
91
92      #オペ看が1人(A)ならばオペ看ペアグループより1人(B) → !A | B
93      s="オペ看が1人(A)ならばオペ看ペアグループより1人(B)"+' '+"day"+str(day) #Keyword +space +variable words
94      sc3.AddSoft(~A |B,s,7) #本来はAddHardでよいが、手書き勤務表が対応していないためにSoftにする
95      #sc3.AddHard(~A|B,s)
```

Python チュートリアル１３　複雑な仕様の例　オペ看０人

■ オペ看が０人(A)ならば、不在時グループからは、２人(B)　です。AならばBは、~A|Bになります。A,B共ブール変数です。　オペ看0人は、ORの否定をとれば、実現できます。
さらに、手術看護師不在時(0人)のときのみ、該当グループ内の優先度が高いスタッフを割り当てるようにしたいので、Aならば、優先スタッフ(B)であるというアサーション=Aでないなら必ずBが成立する　をカウントします。目標値は、優先看護師人数分になり、それに対して足りないだけペナルティとしてカウントされます。現在は、優先看護師が1人ですが、将来2人になってもソースの変更は、必要ありません。

```
98  def オペ看0人(day):
99      ope_nurse_list=オペ看手術シフトリスト取得(day)
100     A=~sc3.Or(ope_nurse_list) #オペ看0人
101
102
103     off_ope_nurse_list=オペ看不在時グループの手術シフトリスト取得(day)
104     B=sc3.SeqExpr(2,2,2,off_ope_nurse_list) #オペ看不在時は、看不在時グループから2人を確保
105     s='オペ看0人'+' '+daydef[day] #Keyword +space +variable words
106     sc3.AddSoft( ~A|B,s,7)#本来はAddHardでよいが、手書き勤務表が対応していないためにSoftにする
107     #sc3.AddHard(~A|B,s)
108     VList=[]
109
110     #s="手術看護師不在時優先"+' '+"day"+str(day)
111     s="手術看護師不在時優先"+' '+daydef[day]#Keyword +space +variable words
112
113     for person in 手術看護師不在時優先:
114         v1=sc3.GetShiftVar(person,day,'手術')
115         #sc3.AddSoft(~A|v1,'',3)#soft count対象
116         VList.append(~A|v1)
117     sc3.AddSoft(sc3.SeqError(len(VList),len(VList),len(VList),VList),s,3)
```

Python チュートリアル１３　互いに背反

■ オペ看の人数2人、1人、0人という事象は、同時には起こりえない事象であり、かつ一つだけ、必ず起きます。従い、これらを独立に記述してよく、かつ、これらを網羅すれば、全てのケースを網羅したことになります。

	手術看護師とペアグループ	手術看護師不在時グループ	優先度
オペ看2人	0	0	高い
オペ看1人	1	0	中
オペ看0人	0	2	低い

```
for day in 今月診療日:
    #sc3.print(str(day)+'\n')
    オペ看2人(day)#オペ看が2人の場合は、オペ看護ペアグループ0  AND  手術看護師不在時グループ0
    オペ看1人オペ看ペアグループより1人(day)    #オペ看が1人の場合は、オペ看ペアグループより1人ペアとする
    オペ看0人(day)        #オペ看が0人の場合は、手術看護師不在時グループより2人  さらにその中でも優先2人が存在
```

このように、複雑な仕様でも、表にして事象の網羅を行うことで、整理して実装することが出来ます。

Python チュートリアル１３　週当たりの勤務回数

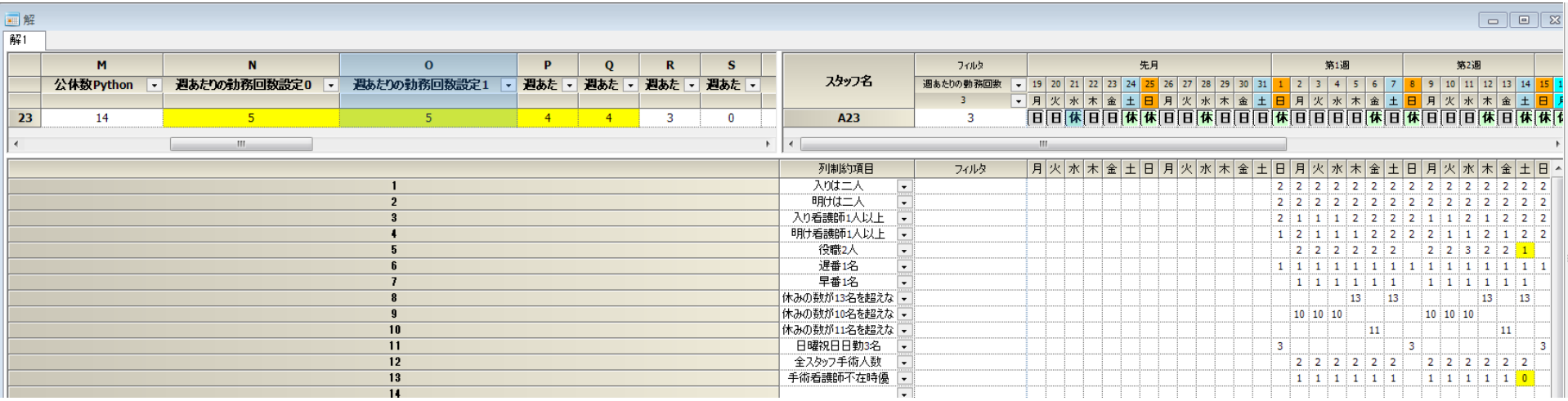
■週当たりの勤務回数を３回（目標値）を行うスタッフが契約上1人います。
■月曜日からカウントし、日曜日が今月にかかるときは、制約します。末尾は、端数になるので、目標値を線形補正します。つまり、7日毎廻して最後は端数処理となります。

```
21  def 週あたりの勤務回数設定():
22  ⟶for person in 週あたりの勤務回数.keys():
23  ⟶⟶n=週あたりの勤務回数[person]
24  ⟶⟶sc3.print(staffdef[person]+'の週当たりの勤務回数は'+str(n)+'回に制約\n')
25  ⟶⟶week=0
26  ⟶⟶for mon in 月:#月曜日を起点に1週間見る
27  ⟶⟶⟶if mon+6 <制約開始日:
28  ⟶⟶⟶⟶continue
29  ⟶⟶⟶sc3.print('月曜日 '+str(mon)+'\n')
30  ⟶⟶⟶VList=[]
31  ⟶⟶⟶for d in range(7):
32  ⟶⟶⟶⟶day=mon+d
33
34  ⟶⟶⟶⟶if day >制約終了日:
35  ⟶⟶⟶⟶⟶break
36  ⟶⟶⟶⟶else:
37  ⟶⟶⟶⟶⟶v1=sc3.GetShiftVar(person,day,'休日集合')#
38  ⟶⟶⟶⟶⟶VList.append(~v1)#休日の否定は勤務
39  ⟶⟶⟶s="週あたりの勤務回数設定"+str(week)+' '+"person"+str(person) #Keyword+ space +variable word
40  ⟶⟶⟶week+=1
41  ⟶⟶⟶if len(VList)==7:
42  ⟶⟶⟶⟶sc3.AddSoft(sc3.SeqError(3,3,4,VList),s,3) #level 3 許容エラー4
43  ⟶⟶⟶else:
44  ⟶⟶⟶⟶nn=int(len(VList)*n/7);#7日に満たない場合は、リニア目標値設定 端数切捨て
45  ⟶⟶⟶⟶sc3.print(str(nn)+"に目標値を設定しました。\n")
46  ⟶⟶⟶⟶sc3.AddSoft(sc3.SeqError(nn,nn,4,VList),s,3)#level 3 許容エラー4
```

Python チュートリアル１３　ＧＵＩ出力

■公休数Python
■週あたりの勤務回数設定０/1/2/3/4/5
■手術看護師不在時優先人数

は、Pythonで記述しましたが、下記の通りGUIで計数されています。目標値から逸脱している箇所については黄色で表示されます。



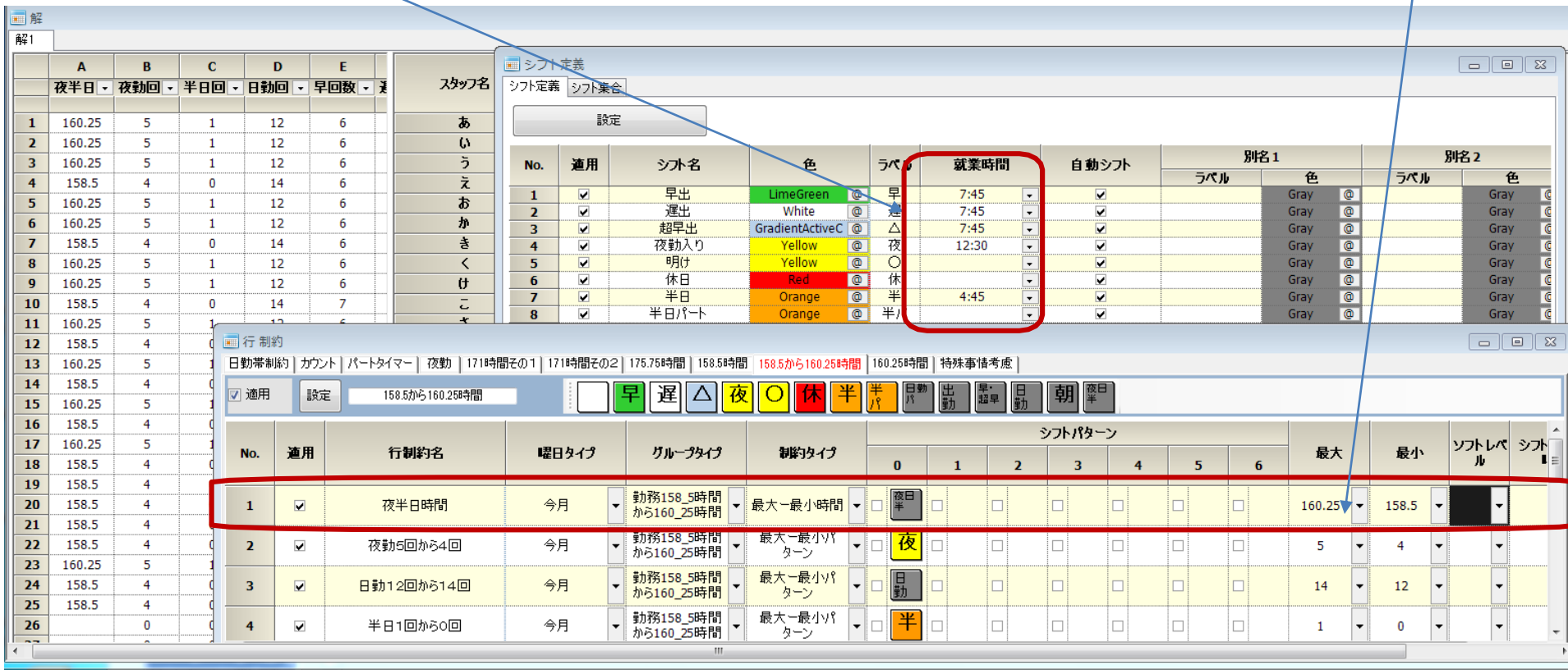
	M	N	O	P	Q	R	S
	公休数Python	週あたりの勤務回数設定0	週あたりの勤務回数設定1	週あた	週あた	週あた	週あた
23	14	5	5	4	4	3	0

スタッフ名	フィルタ	先月													第1週							第2週							
	週あたりの勤務回数	19	20	21	22	23	24	25	26	27	28	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	3	月	火	水	木	金	土	日	月	火	水	木	金	土	日	月	火	水	木	金	土	日	月	火	水	木	金	土	日
A23	3	日	日	休	日	日	休	休	日	日	休	日	日	日	休	日	日	日	日	休	日	休	日	日	日	休	日	休	休

	列制約項目	フィルタ	月	火	水	木	金	土	日	月	火	水	木	金	土	日	月	火	水	木	金	土	日	月	火	水	木	金	土	日
1	入りは二人															2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	明けは二人															2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	入り看護師1人以上															2	1	1	1	2	2	2	2	1	1	2	1	2	2	2
4	明け看護師1人以上															1	2	1	1	1	2	2	2	2	1	1	2	1	2	2
5	役職2人																2	2	2	2	2	2		2	2	3	2	2	1	
6	遅番1名															1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	早番1名																1	1	1	1	1	1		1	1	1	1	1	1	
8	休みの数が13名を超えな																			13		13					13		13	
9	休みの数が10名を超えな																10	10	10					10	10	10				
10	休みの数が11名を超えな																				11							11		
11	日曜祝日日勤3名															3							3							3
12	全スタッフ手術人数																2	2	2	2	2	2		2	2	2	2	2	2	
13	手術看護師不在時優																1	1	1	1	1	1		1	1	1	1	1	0	
14																														

特養時間制約を使わない記述例

■特養のスタッフ勤務時間を極限まで平準化したサンプルです。158.5-160.25時間に制約しています。
■この時間制約を使わないで、記述する方法について述べます。



全スタッフの勤務時間が同じにできないのはなぜ？

■夜勤4人が毎日必要になります。つまり、各スタッフの夜勤回数総和は4の倍数に一致しなければなりません。ここに端数が生じることにより夜勤4回または、5回のスタッフが出現する要請が生じるわけです。そうすると、1夜勤=12.5hの差が生じるのは仕方ないように思えます。

解1

	A 夜半日	B 夜勤回	C 半日回	D 日勤回	E 早回数
1	160.25	5	1	12	6
2	160.25	5	1	12	
3	160.25	5	1	12	
4	158.5	4	0	14	
5	160.25	5	1	12	
6	160.25	5	1	12	
7	158.5	4	0	14	
8	160.25	5	1	12	
9	160.25	5	1	12	
10	158.5	4	0	14	
11	160.25	5	1	12	
12	158.5	4	0	14	
13	160.25	5	1	12	
14	158.5	4	0	14	
15	160.25	5	1	12	
16	158.5	4	0	14	
17	160.25	5	1	12	
18	158.5	4	0	14	
19	158.5	4	0	14	
20	158.5	4	0	14	
21	158.5	4	0	14	
22	158.5	4	0	14	
23	160.25	5	1	12	
24	158.5	4	0	14	
25	158.5	4	0	14	
26		0	0	20	

スタッフ名：あ

列 制約

列制約グループ1 | 列制約グループ2

適用　設定　列制約グループ1

No.	適用	列制約名	曜日タイプ	グループタイプ	シフトタイプ	制約タイプ	最大	最小	異重み時のソフトレベル最大	ソフトレベル
1	☑	超早は1人	今月	全スタッフ	超早出	最大－最小スタッ	1	1		
2	☑	夜勤は4人	今月	全スタッフ	夜勤入り	最大－最小スタッ	4	4	5	5
3	☑	日勤者数	今月	全スタッフ	日勤パート含	最大－最小スタッ	16	14		
4	☑	増員日	増員日	全スタッフ	日勤パート含	最大－最小スタッ		16		
5	☑	風呂日	風呂日	A	日勤パート含	最大－最小スタッ		3		
6	☑	行事日	行事日	リーダー	日勤パート含	最大－最小スタッ		2		
7	☑	夜勤超早男性	今月	男	朝時間	最大－最小スタッ	4	2		
8	☑	夜Aクラス	今月	A	夜勤入り	最大－最小スタッ		1		
9	☑	夜勤超早新人は2人まで	今月	朝時間新人	朝時間	最大－最小スタッ	2			
10	☐	夜勤Aクラスは2人まで	今月	夜勤リーダー	夜勤入り	最大－最小スタッ	2			
11	☑	早出	今月	班あり	早出	最大－最小スタッ		5		
12	☑	早出Aクラス	今月	A	早出	最大－最小スタッ		1		
13	☑	早出ABクラス	今月	AB	早出	最大－最小スタッ		3		
14	☑	早出性別	今月	男	早出	最大－最小スタッ		1		
15	☑	遅出	今月	全スタッフ	遅出	最大－最小スタッ		7		
16	☑	遅出Aクラス	今月	A	遅出	最大－最小スタッ		1		
17	☑	遅出ABクラス	今月	AB	遅出	最大－最小スタッ		3		
18	☐	品質	品質	品質	早出	最大－最小スタッ		8		
19	☐	拘束	拘束	拘束	早出	最大－最小スタッ		6		
20	☐	事故	事故	事故	早出	最大－最小スタッ		2		

勤務時間の平準化案

■しかし、160.25hと158.5ｈは、夜勤回数、日勤回数、半日回数を各々下の組み合わせのように制約出来れば、時間制約を使わずに済みます。下の組みわせのいずれかを満たすという制約を書けばよいことが分かります。

行 制約

日勤帯制約 | カウント | パートタイマー | 夜勤 | 171時間その1 | 171時間その2 | 175.75時間 | 158.5時間 | 158.5から160.25時間 | 160.25時間 | 特殊事情考慮

適用 | 設定 | 160.25時間 | 早 | 遅 | △ | 夜 | ○ | 休 | 半 | 半パ | 日勤パ | 出勤 | 早・超早 | 日勤 | 朝 | 夜日半

No.	適用	行制約名	曜日タイプ	グループタイプ	制約タイプ	シフトパターン 0	1	2	3	4	5	6	最大	最小	ソフトレベル	シフトパターン曜日タイ
1	☐	夜勤5回	今月	勤務160_25時間	最大ー最小パターン	夜							5	5		
2	☐	日勤12回	今月	勤務160_25時間	最大ー最小パターン	日勤							12	12		
3	☐	半日1回	今月	勤務160_25時間	最大ー最小パターン	半							1	1		
4	☑	夜半日時間	今月	勤務160_25時間	最大ー最小時間	夜日半							160.25	160.25		

行 制約

日勤帯制約 | カウント | パートタイマー | 夜勤 | 171時間その1 | 171時間その2 | 175.75時間 | 158.5時間 | 158.5から160.25時間 | 160.25時間 | 特殊事情考慮

適用 | 設定 | 158.5時間 | 早 | 遅 | △ | 夜 | ○ | 休 | 半 | 半パ | 日勤パ | 出勤 | 早・超早 | 日勤 | 朝 | 夜日半

No.	適用	行制約名	曜日タイプ	グループタイプ	制約タイプ	シフトパターン 0	1	2	3	4	5	6	最大	最小	ソフトレベル	シフトパターン曜日タイ
1	☐	夜勤4回	今月	勤務158_5時間	最大ー最小パターン	夜							4	4		
2	☐	日勤14回	今月	勤務158_5時間	最大ー最小パターン	日勤							14	14		
3	☐	半日0回	今月	勤務158_5時間	最大ー最小パターン	半							0	0		
4	☑	夜半日時間	今月	勤務158_5時間	最大ー最小時間	夜日半							158.5	158.5		

Pythonで制約記述

■158.5hの場合組み合わせのAndと、160.25hの組み合わせのAndのOrを取ればよいです。

■158.5hから160.25hのスタッフプロパティを持つスタッフについて(line31)上記制約を適用すればよいです。

■時間制約を使わない記述方法について述べました。

```
1  import sc3
2
3  def 勤務時間設定sub(person):
4      str=staffdef[person]+'の勤務を158.5時間から160.25時間の範囲内に制約'
5      sc3.print(str+'します。\n')
6      VList半日=[]
7      VList夜=[]
8      VList日勤=[]
9      for day in 全日:
10         if day<制約開始日:
11             continue
12         v1=sc3.GetShiftVar(person,day,'半日')
13         VList半日.append(v1)
14
15         v1=sc3.GetShiftVar(person,day,'日勤')
16         VList日勤.append(v1)
17
18         v1=sc3.GetShiftVar(person,day,'夜勤入り')
19         VList夜.append(v1)
20     半日1=sc3.SeqExpr(1,1,2,VList半日)
21     半日0=sc3.SeqExpr(0,0,2,VList半日)
22     夜4=sc3.SeqExpr(4,4,2,VList夜)
23     夜5=sc3.SeqExpr(5,5,2,VList夜)
24     日12=sc3.SeqExpr(12,12,2,VList日勤)
25     日14=sc3.SeqExpr(14,14,2,VList日勤)
26     勤務158_5時間=日14 & 夜4 & 半日0
27     勤務160_25時間=日12 & 夜5 & 半日1
28     sc3.AddHard(勤務158_5時間 | 勤務160_25時間,str)
29
30 def 勤務時間設定():
31     for person in 勤務158_5時間から160_25時間:
32         勤務時間設定sub(person)
33
34 勤務時間設定()
```

フェーズ・タスクモードでの記述

■GetShiftVarの代わりにGetTaskVarを使います。

■右は、プロジェクトサンプルsudoku_task.nurse3の記述です。

■GetTaskVarの書式は、GetTaskVar(person,day,phase,task)です。

Pythonによる制約

設定

```
import sc3
for day in 全日: #列制約
    for ph in range(3):
        for タスク in 全タスク:
            s='PythonColumnConstraint_'+daydef[day]+' '+タスク +'\n'
            sc3.print(s)
            V=[]
            for 人 in 全スタッフ:
                V.append(sc3.GetTaskVar(人,day,ph,タスク))
            sc3.AddHard(sc3.SeqLE(1,1,V),s)

for 人 in 全スタッフ:#行制約
    for タスク in 全タスク:
        V=[]
        s='Python_Row_Constraint_'+staffdef[人]+' '+タスク+'\n'
        for day in 全日:
            for ph in range(3):
                V.append(sc3.GetTaskVar(人,day,ph,タスク))
        sc3.AddHard(sc3.SeqLE(1,1,V),s)

for 人 in スタッフブロックトップ: #ブロック制約
    for day in 全日:
        for タスク in 全タスク:
            V=[]
            s='Python_Block_Constraint_'+staffdef[人]+' '+daydef[day]+' '+タスク+'\n'
            for i in range(3):
                for j in range(3):
                    V.append(sc3.GetTaskVar(人+i,day,j,タスク))
            sc3.AddHard(sc3.SeqLE(1,1,V),s)
```

NRPのサンプル　その１

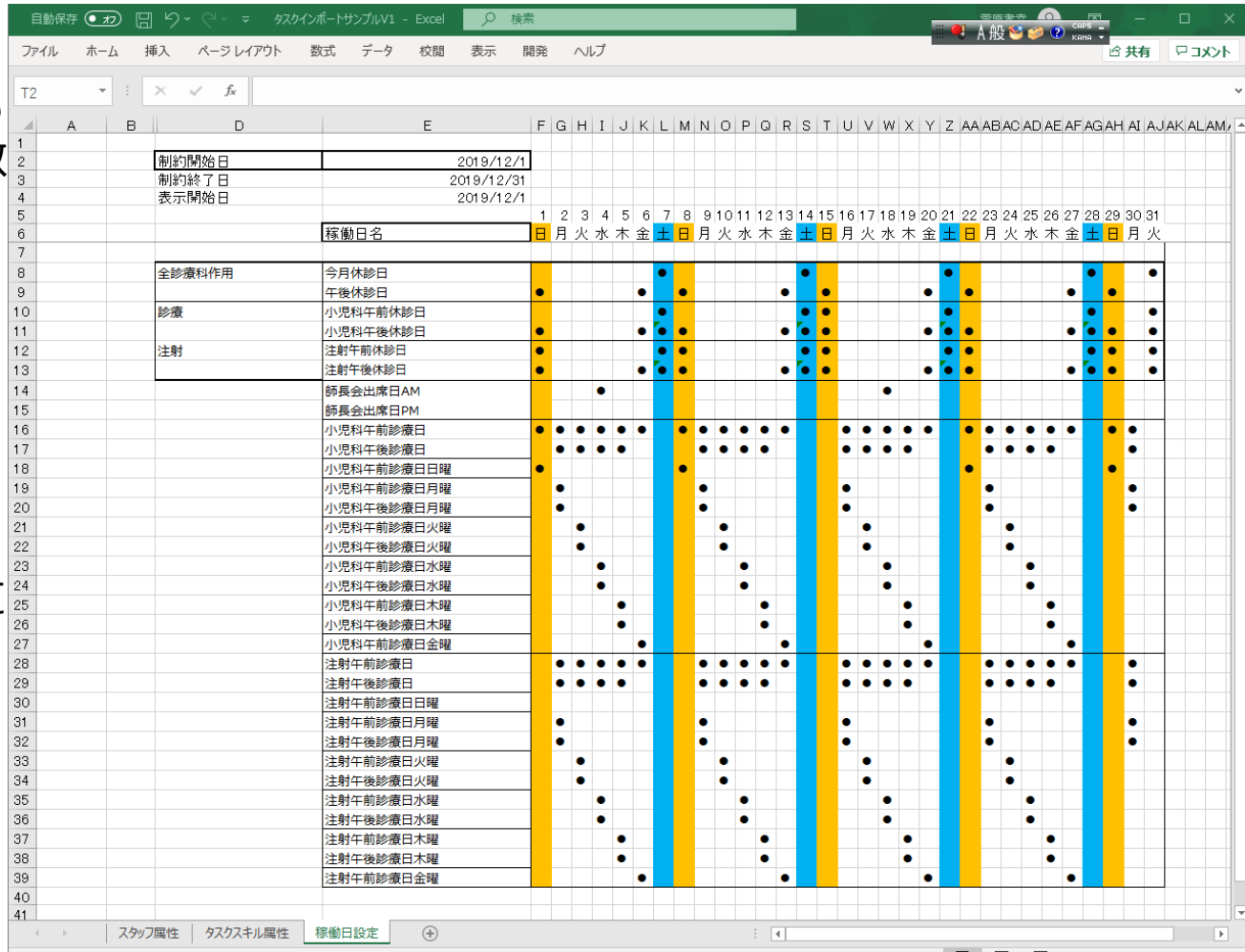
■タスクインポートサンプルV1.nurse3をプロジェクトサンプルとして解説します。
■この問題は、AMとPMにナース・アシスタントを診療科に必要人数を割り当てる問題です。
■診療科では、休診日がExcelで定義されています。
■ナース・アシスタントは、各々スキルの属性がありExcelで定義されています
■各スタッフは、出勤属性がありExcelで定義されています。
■サンプルでは、診療科休診日、タスクスキル属性、稼働日設定のExcelシートを読み込んでいます。
■Pythonでは、各スタッフが、曜日AM/PM毎と月毎に出勤の可否の属性を読んで、診療科への割り当て(支援業務）を禁止することを記述しています。
■割り当て禁止は、余タスクに強制することで実現します。余タスクが割りあえてられたら、どの科にも配置されない、ということを意味します。従い、各科での必要人数をスタッフの、年休、スキル、出勤属性を考慮しながら配置するという問題になります。

NRPのサンプル　稼働日設定シート

■稼働日設定シートです。
ユーザ入力は、休診日のみですが、Excelシートでは各曜日の診療日をExcelで演算させています。SC3内でも集合演算は可能ですが、多数のエントリを要するので、Excelで記述しています。

■必要人数は、各曜日AM・PMの各科診療日について制約する上で必要です。

■一方休診日の情報も必要になります。
休診日は、タスクではなく、シフトで設定しています。

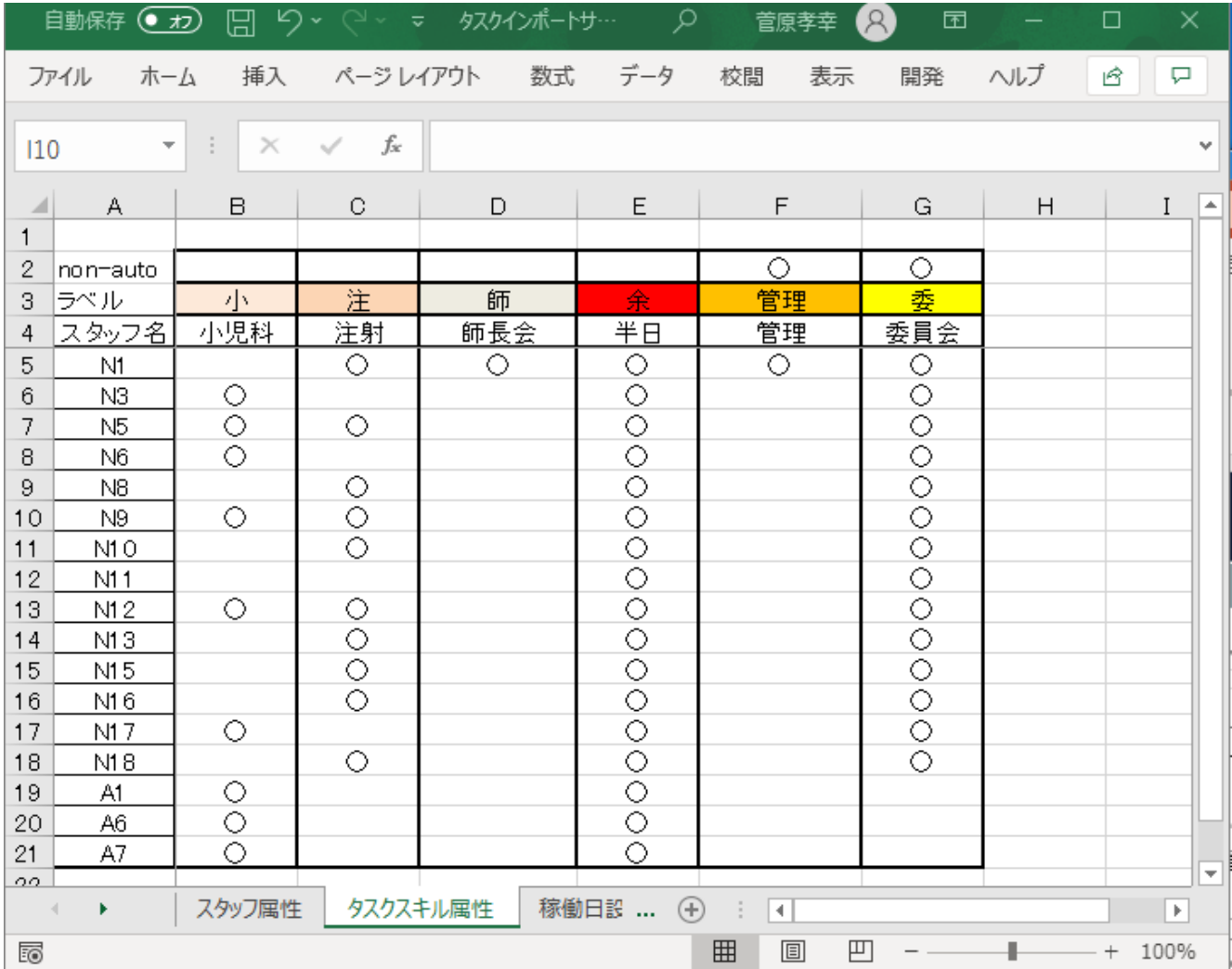


NRPのサンプル　タスクスキル属性シート

■タスク種類とスタッフ毎のスキル属性シートになります。

■スキル能力は、月々で更新されるのでExcelで管理します。

■non-auto属性は、予定の入力のみに有効なタスクという意味です。通常タスクは、オンとオフ両方の状態を常に意識で記述する必要がありますが、これが○だと、予定入力に使用を限定していることで、その注意は必要ありません。ただし、Python記述において使用することはできないので注意してください。



	A	B	C	D	E	F	G
1							
2	non-auto					○	○
3	ラベル	小	注	師	余	管理	委
4	スタッフ名	小児科	注射	師長会	半日	管理	委員会
5	N1		○	○	○	○	○
6	N3	○			○		○
7	N5	○	○		○		○
8	N6	○			○		○
9	N8		○		○		○
10	N9	○	○		○		○
11	N10		○		○		○
12	N11				○		○
13	N12	○	○		○		○
14	N13		○		○		○
15	N15		○		○		○
16	N16		○		○		○
17	N17	○			○		○
18	N18		○		○		○
19	A1	○			○		
20	A6	○			○		
21	A7	○			○		

NRPのサンプル　スタッフ属性シート

■各スタッフの曜日出勤属性が記述されています。
■Pythonでは、これらのプロパティを辞書として読み込む必要があるので、数字0で、記述します。
。
■余り表示属性は、余りの最小化を行わないという意味です。クリアにすると余りタスクにペナルティが与えられて、余り人員の最小化が行われるPython記述としています。

タスクインポートサンプルV1 - Excel

スタッフ名	コメント	全スタッフ属性	職能	余表示属性	日曜出勤属性	月AM出勤属性	月PM出勤属性	火AM出勤属性	火PM出勤属性	水AM出勤属性	水PM出勤属性	木AM出勤属性	木PM出勤属性	金AM出勤属性	出勤1月属性	出勤2月属性	出勤3月属性	出勤4月属性	出勤5月属性	出勤6月属性	出勤7月属性	出勤8月属性	出勤9月属性	出勤10月属性	出勤11月属性	出勤12月属性
N1	研修	全スタッフ	ナース	全表示																						
N3		全スタッフ	ナース	全表示																						
N5		全スタッフ	ナース	全表示																						
N6		全スタッフ	ナース	全表示																						
N8		全スタッフ	ナース	全表示																						
N9		全スタッフ	ナース	全表示																						
N10		全スタッフ	ナース	全表示																						
N11		全スタッフ	ナース	全表示																						
N12		全スタッフ	ナース	全表示			0				0															
N13		全スタッフ	ナース	全表示		0	0		0		0	0	0													
N15		全スタッフ	ナース	全表示						0	0															
N16		全スタッフ	ナース	全表示				0	0																	
N17		全スタッフ	ナース	全表示						0	0															
N18		全スタッフ	ナース	全表示	0	0	0		0	0	0	0	0	0												
A1		全スタッフ	アシスタント	全表示																						
A6		全スタッフ	アシスタント	全表示																						
A7		全スタッフ	アシスタント	全表示	0						0															

スタッフ属性　タスクスキル属性　稼働日設定

NRPのサンプル　python記述のエッセンスその１

■支援禁止とするためのサブルーチンになります。
■シフトとタスク混在の制約になるので、Pythonで制約します。
■日勤(1日出勤シフト）または、am(AMのみ出勤シフト）だったら　余りタスクにするという　制約になります。それ以外のシフトでは、フェーズ定義により自動的にタスクは存在しないので、記述する必要はありません。
■AならBという記述になりますので、A→B＝！A｜B　ですので、次のような記述になります。フェーズAMとPMに分けて、以下のような制約記述となります。

```
def 支援禁止(person,day,phase):#日勤なら余りを強制する
    if phase==AM:
        sam='日勤ならAM余りのみ'+staffdef[person]+' '+daydef[day]
        Sam=sc3.GetShiftVar(person,day,'日勤') | sc3.GetShiftVar(person,day,'am')
        vam=sc3.GetTaskVar(person,day,AM,'半日')#余り
        #A->B =!A|B
        sc3.AddHard(~Sam|vam,sam)
    else:
        spm='日勤ならPM余りのみ'+staffdef[person]+' '+daydef[day]
        Spm=sc3.GetShiftVar(person,day,'日勤')
        vpm=sc3.GetTaskVar(person,day,PM,'半日')#余り
        #A->B =!A|B
        sc3.AddHard(~Spm|vpm,spm)
```

NRPのサンプル　python記述のエッセンスその２

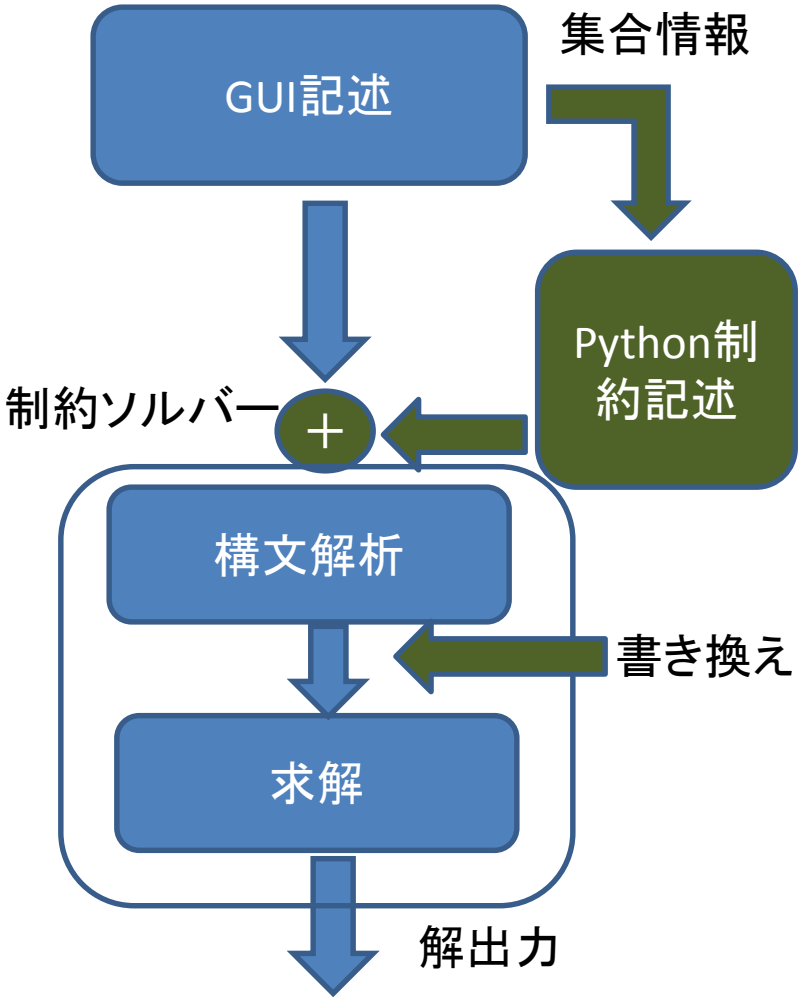
■余り人員を最小化するための記述です。
■余表示者でないスタッフのみに制約します。
■半日タスク(余りタスク)にソフトペナルティ(ソフトレベル６）を与えます。
■後は、ソルバが総和重みが最小となるように、勝手にアサインしてくれます。

```
def 余り最小化():
    for person in 全スタッフ:
        if person not in 余表示:
            for day in 今月稼動日:
                vam=sc3.GetTaskVar(person,day,0,'半日')
                s=staffdef[person]+daydef[day]+"AM"
                sc3.AddSoft(~vam,s,6) #余表示者でなければ、余TASKは、できるだけ少なくする
                vpm=sc3.GetTaskVar(person,day,1,'半日')
                s=staffdef[person]+daydef[day]+"PM"
                sc3.AddSoft(~vpm,s,6) #余表示者でなければ、余TASKは、できるだけ少なくする
```

制約のコントロール

■年末年始等の長期休暇等により、その月だけ制約を別にしたい場合があります。プロジェクトファイルを別にするやり方もありますが、pythonを使って、一つのプロジェクト上で、GUIのグループまたは個別の制約をオンオフすることも可能です。また、人が介在することなく自立的にオンオフする設計も可能です。

■GUI制約のオンオフが起きるタイミングは、右図制約エンジン(制約ソルバー)内で、構文解析が終わった直後です。
■従い、GUIは制約を書き換えたことは分かりません。解表示の黄赤色表示は、GUIが認識していたソフトレベルで行われます。



制約グループのオンオフ

■下のように列・ペア・行共に制約はグループ単位でオンオフすることができます。

制約グループのオンオフ

■下のように列・ペア・行共に制約はグループ単位でオンオフすることができます。（プロジェクトサンプル　深夜公休1回.nurse3参照)

■前ページで正循環のグループをオンするには、

```
sc3.ConstraintEnable('正循環’)
```

■深準パターンの制約グループをオフするには、

```
sc3.ConstraintEnable('深準パターン',False)
```

と記述します。

制約のオンオフ

■制約をオンオフするには、制約名を'制約グループ名　ドット　制約名'で指定します。

■たとえば、深準パターングループの夜勤3連不可制約をオンするには、

```
sc3.ConstraintEnable('深準パターン.夜勤3連不可',True)
```

■オフするには、

```
sc3.ConstraintEnable('深準パターン.夜勤3連不可',False)
```

と記述します。

■ただし、制約をオンするには、その上位の制約グループがオンに設定されていることが必要です。記述の順序は問いません。

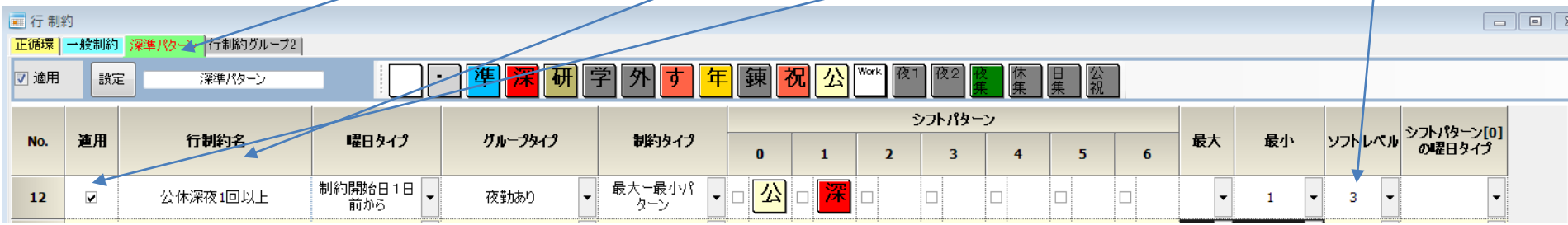
■オンオフは、揮発性です。プロジェクトファイルには保存されません。

制約のソフトレベルの変更

■制約のソフトレベルを変更するには、制約名を'制約グループ名　ドット　制約名'で指定します。

■たとえば、次は、ソフトレベルを３→４にする記述です。

```
sc3.ConstraintEnable('深準パターン.公休深夜１回以上',True,4)
```



■ソフトレベルを変更した場合、ソルバーにオンオフ状態、重み、許容エラー数の設定を行う必要があります。(SetSoftLevelの項をご参照ください。)

■ソフト制約からハード制約に変更する場合は、ソフトレベル0　にします。

制約のmin max値の変更

■制約のmin/maxを変更するには、制約名を'制約グループ名　ドット　制約名'で指定します。

■たとえば、次は、min>=1をmin>=2に変更する記述です

```
sc3.SetMinMax('深準パターン.公休深夜1回以上',True,2,-1)
```

■上の場合、記述しない場合は、-1を指定します。上の場合max値は、-1であり指示は無視せよ、という意味になります。

ソフト重み・許容エラー数の変更

■ソフトレベルを変更すると、そのソフトレベルの適用可否・重み・許容エラー数を設定しなければなりません。

■例えば、GUI上は、下のようになっていても、前頁のようにソフトレベルを書き換えた場合、対応する記述が無くなってしまうので、ソルバーに教えてあげなくてはいけません。これを怠ると制約が適用されなくなります。

sc3.SetSoftLevel(4,‘row’,True,444,3)

■‘row’行　‘column’列　‘pair’ペア　‘planned‘予定　に対応します。

■ソフト制約をハード制約に変更した場合は、記述不要です。

ソルバ設定の変更

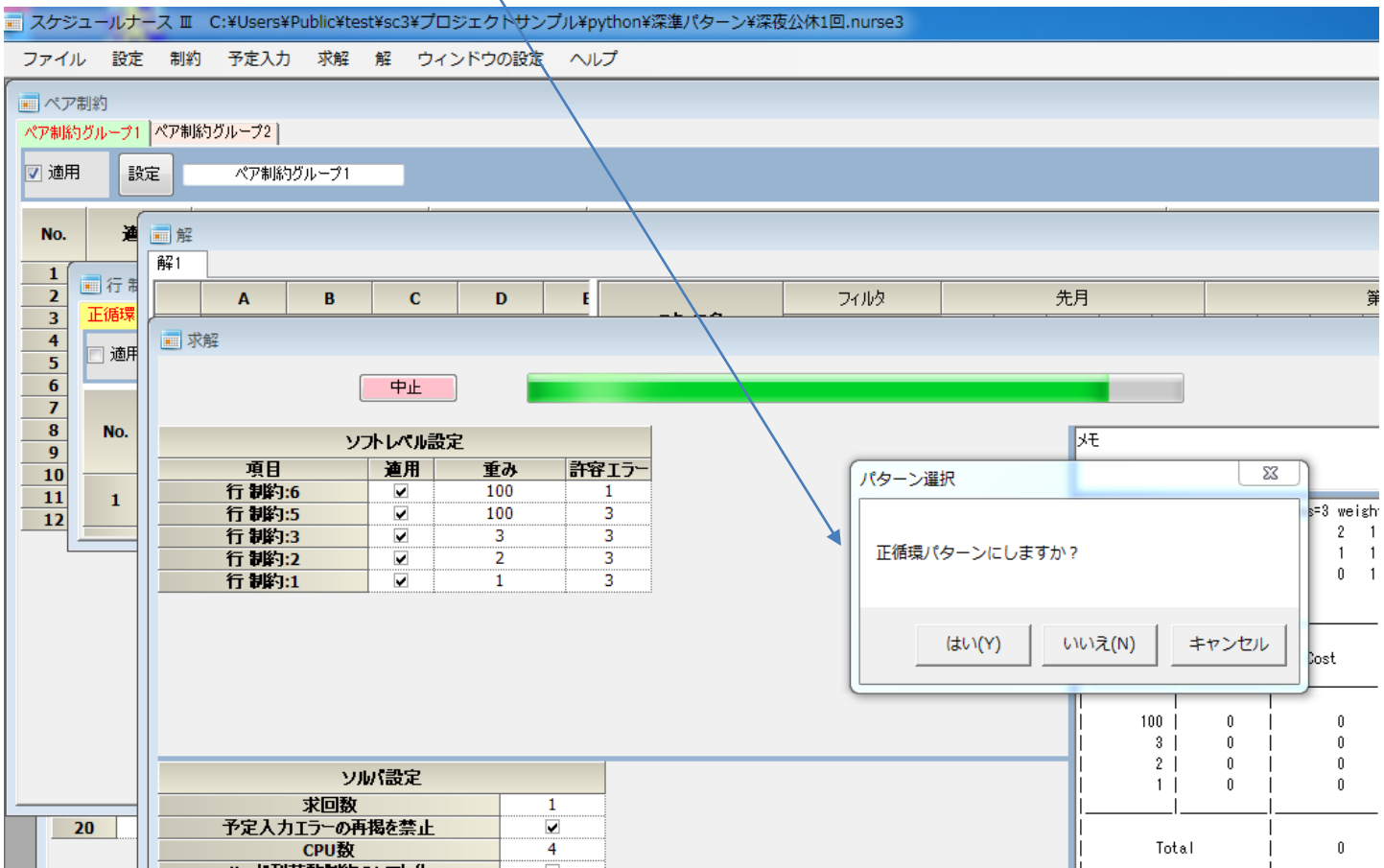
■下のような形式で、設定の変更が可能です。全て数字でDisableは0としてください。sc3.SetSolvingParameter('implict_hard_to_soft_enable',1)

	ソルバ設定	
N_of_solutions	求回数	1
not_repeat_planned_errors	予定入力エラーの再掲を禁止	☑
NofCpus	CPU数	4
implict_hard_to_soft_enable	ハード列基数制約のソフト化	☑
HardTimeout	ハードタイムアウト(秒)	10
ErrorAnalysis	エラー解析	☑
SoftTimeout	ソフトタイムアウト(秒)	15
UseExternalConstraint	言語制約の使用	☑
BestEffotsTimeout	エラー解析時のソフトタイムアウト(秒)	5
ShowUnsatisfactoryHardItems	エラー解析時予定入力を肯定	☐
Algorithm	アルゴリズム変更(ここをクリック)	1
final_soft_timeout	最終ソフトタイムアウト	10

制約のコントロールデモ

■Win32のAPIを呼び出すことが可能です。
(プロジェクトサンプル　深夜公休1回.nurse3参照)

ただし、将来予定されるネットエンジン化Versionでは使用できません。
Windows Versionのみです。)



Python 制約関数まとめ

■ 詳細は、説明ページをご覧ください

	書式	内容	使用箇所	戻り値
SeqLE	ＳｅｑＬE(min,max,List)	ハード基数制約	AddHard内のみ	
SeqError	SeqError(min,max,allowable erros,List)	ソフト基数制約	AddSoft内のみ	
SeqComp	SeqComp(ListA,ListB)	ΣListA==ΣListB		1 bit
SeqExpr	SeqExpr(min,max,Type、List)			1 bit

用語集

ソルバ	答えを出す脳、に相当するソフトウェア(求解エンジン)
制約	勤務のルール
解	制約を満たす答え
ハード制約	必ず満たさないといけない制約。
ソフト制約	出来れば満たしたい制約
オプティマイズ	最適化。エラーの数を1個づつ減らして最小化すること。
リソース	人的資源。使用例）リソースがない➡人がいない
行	横方向の列
列	縦方向の列
充足	制約を満たすこと
エラー	失敗。誤差（目標値からのずれ、ペナルティ）
ボトルネック	その制約を満たすことが難しいために解がない状態
トレードオフ	何かを達成するために、別な何かを犠牲にしなければならないこと

用語集2

プロジェクトファイル	制約,予定入力や解を収めたお客様専用設定ファイル。通常月毎に名前を変えて作成します。
SE	システムエンジニア
ハードエラー	解が存在しないこと（ソルバが矛盾を認識したときに出すエラー）
solution1	1番目の答え、solution2は、2番目の答え