スターターキットですぐに試せる

鉄道模型シミュレーター

時刻表通り自動運行

Python 3.7.8 Shell

File Edit Shell Debug Options Window Help

```
catch 86 40 1 1
2020-07-26 12:58:20.997251: [駅２]から[B0040]発車
catch 87 40 1 1
catch 79 40 1 1 sequence1ii
catch 82 40 1 1
2020-07-26 12:59:00.146719: [駅１]から[A0200]発車
catch 81 39 1 1 sequence1Ao
catch 86 39 1 1
catch 87 39 1 1
catch 79 39 1 1 sequence1ii
2020-07-26 12:59:40.317842: [駅１]から[B0240]発車
catch 83 40 1 1 sequence1Bo
catch 80 39 1 1
catch 86 40 1 1
2020-07-26 13:00:20.511538: [駅２]から[B0240]発車
```

```
def sequence1ii(train):
    if train == train1:
        point1.SetBranch(0)
    elif train == train2:
        point1.SetBranch(1)

def sequence1Ao(train):
    point2.SetBranch(0)

def sequence1Bo(train):
    point2.SetBranch(1)

ats1ii.forward = sequence1ii
ats1Ao.forward = sequence1Ao
ats1Bo.forward = sequence1Bo
```

列車番号	駅１	駅２	駅１	駅２
A0000	00:00	00:40	01:10	
B0040	00:40	01:20	01:50	
A0200	02:00	---->	03:00	
B0240	02:40	03:20	03:50	

スターターキットですぐに試せる
鉄道模型シミュレーター
時刻表通り自動運行

rapidnack – 著

・以下のサイトで、本書で使用した TCP サーバー機能を追加したレイアウトファイル、モジュール、スクリプトを入手できます。

https://github.com/Rapidnack/ATOVRMNX

・本書中の会社名や商品名は、該当する各社の商標または登録商標です。本書中では TM および ® マークは省略させていただいております。

はじめに

「鉄道模型シミュレーター」のスクリプトが Python に変更され、しかも無料のスターターキットが提供されていました。

http://www.imagic.co.jp/hobby/products/vrmnx/

滑らかに走る列車を眺めているだけでも癒されますが、より楽しむために「時刻表どおりに複数の列車を運行させる Python スクリプト」とそれに必要な Python モジュールを作りました。

次のようなフォーマットの時刻表の文字列を解釈し、複数の列車を時刻表どおりに運行します。列車番号ごとに各駅の発車時刻 (分：秒) をカンマで区切って並べています。各行の左端の時刻が始発駅の発車時刻で、右端の時刻が終着駅に到着時刻です。時刻以外 (---->) が書かれている駅は通過します。

```
列車番号,駅１   ,駅２   ,駅１   ,駅２
A0000   ,00:00  ,00:40  ,01:10
B0040   ,00:40  ,01:20  ,01:50
A0200   ,02:00  ,----->  ,03:00
B0240   ,02:40  ,03:20  ,03:50
```

```
列車番号,駅１   ,駅２   ,駅３   ,駅４   ,駅１   ,駅２   ,駅３   ,駅４
A0000   ,00:00  ,00:40  ,01:20  ,02:00  ,02:30
B0000   ,       ,00:00  ,00:40  ,01:20  ,02:00  ,02:30
C0000   ,       ,       ,00:00  ,00:40  ,01:20  ,02:00  ,02:30
D0000   ,       ,       ,       ,00:00  ,00:40  ,01:20  ,02:00  ,02:30
```

「鉄道模型シミュレーター」に TCP サーバー機能を追加して、外部の自動運転用 Python スクリプトからコマンドを送ります。

TCP サーバー機能を追加したレイアウトファイル、モジュール、スクリプトを GitHub で公開しています。

https://github.com/Rapidnack/ATOVRMNX

GitHub で公開しているレイアウトファイルは、線路を配置しただけの殺風景なものです。「鉄道模型シミュレーター」初心者のため背景まで手が出ませんでした。

2020 年 7 月 rapidnack

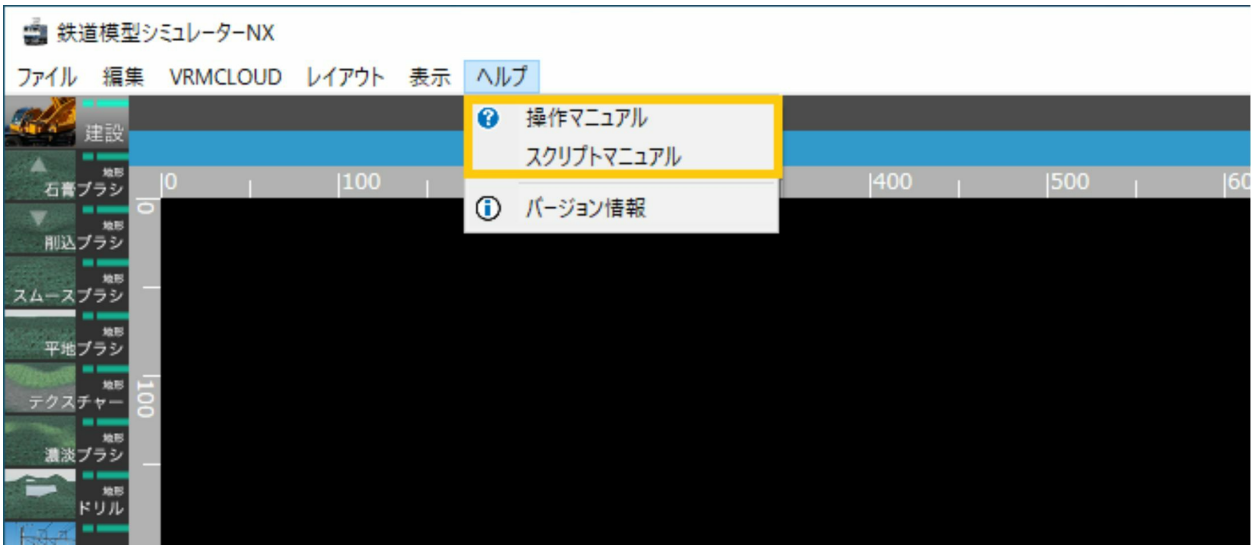
1　　鉄道模型シミュレータースターターキットをインストール

こちらからインストーラをダウンロードします。
http://www.imagic.co.jp/hobby/products/vrmnx/start/starter/

Web ページの説明に従って「鉄道模型シミュレーター NX 」（以下 VRMNX と称する）をインストールします。

VRMNX を起動します。新しいバージョンが在る場合は画面の指示に従いバージョンアップします。新しいインストーラがダウンロードされて起動するまで数分掛かります。

ヘルプメニューから「操作マニュアル」と「スクリプトマニュアル」の Web ページを表示できます。

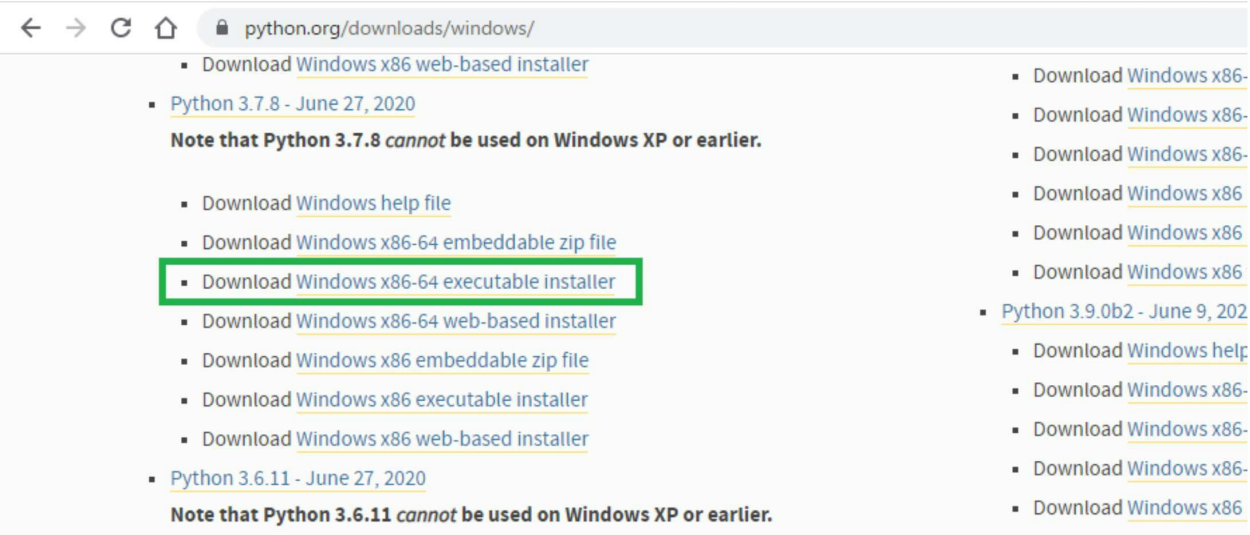


このバージョンで動作確認しました。

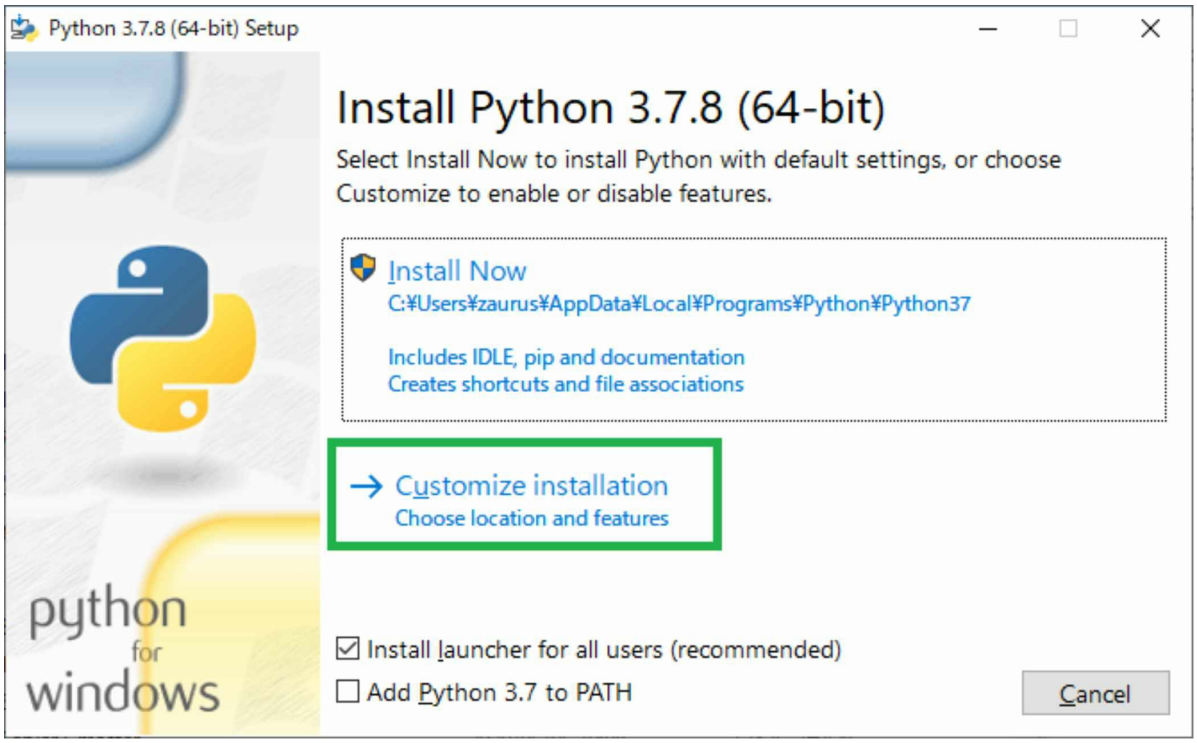
2　Python 3.7 をインストール

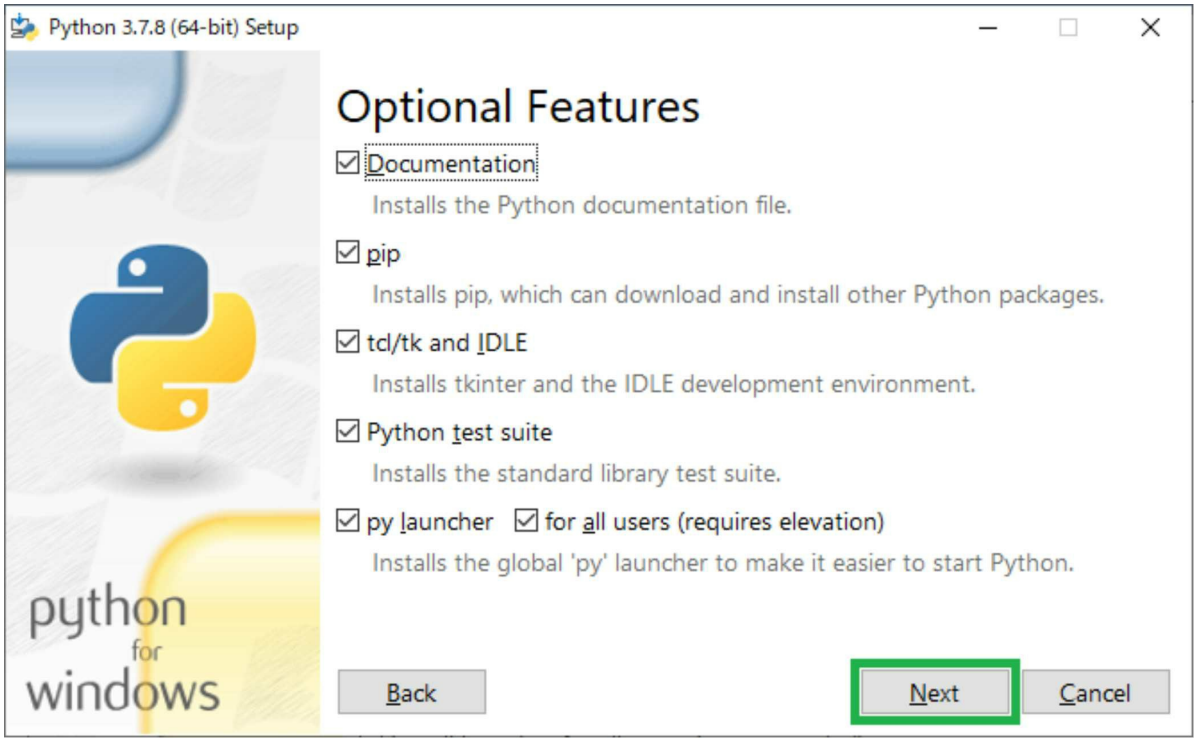
VRMNX に TCP サーバー機能を追加するために Python の socket パッケージを使います。VRMNX の Lib フォルダに入っていないので、通常の Python を PC にインストールします。

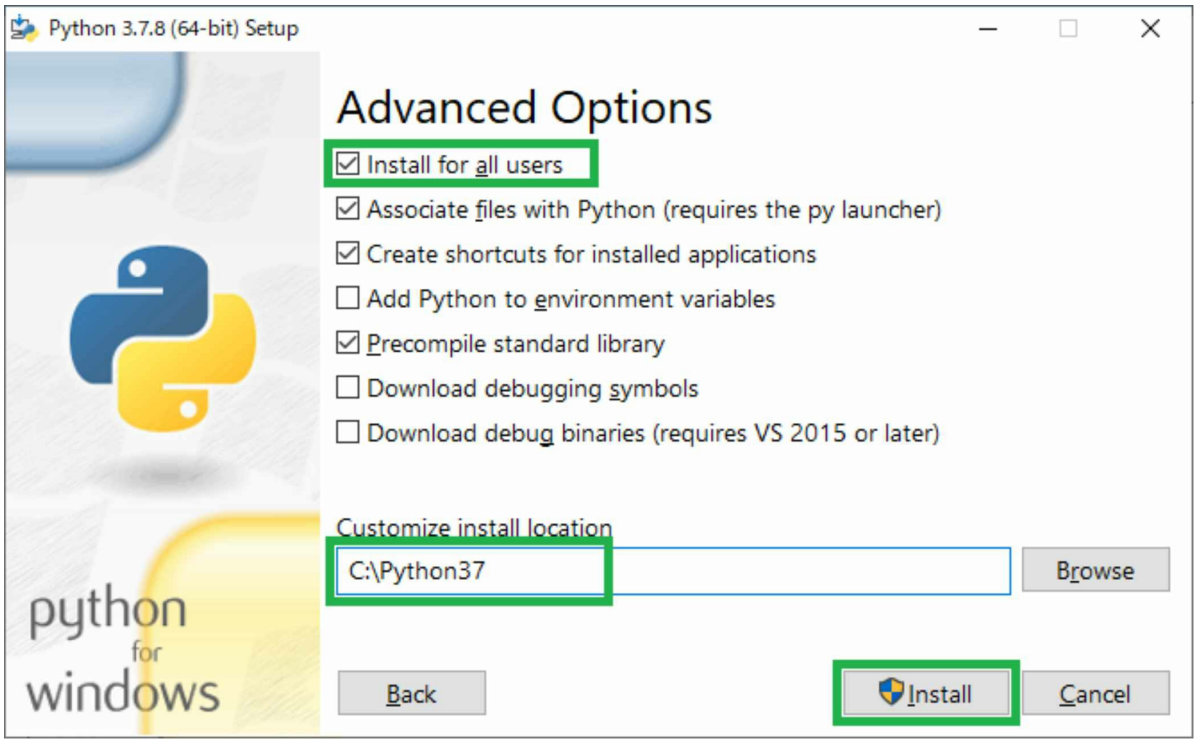
VRMNX に合わせて、Python3.7 のインストーラをダウンロードします。
https://www.python.org/downloads/windows/

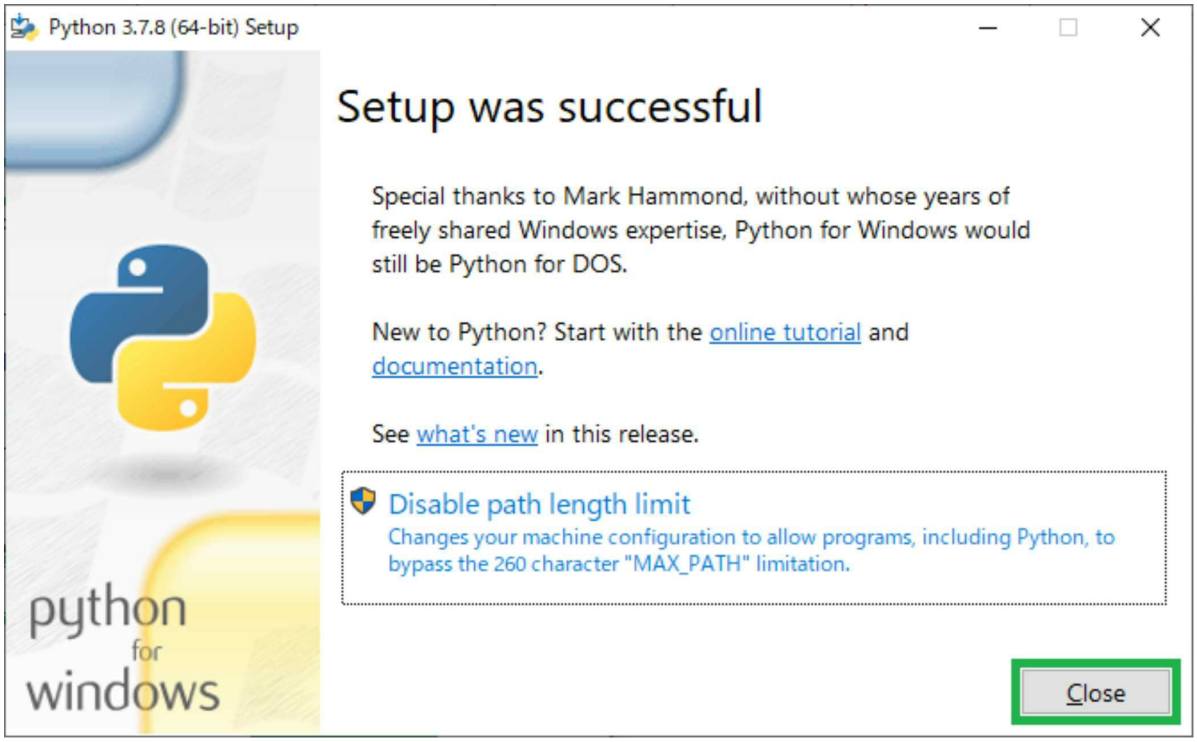


ダウンロードしたインストーラを実行してインストールします。
「 python-3.7.8-amd64.exe 」


Python 3.7.8 (64-bit) Setup
Install Python 3.7.8 (64-bit)
Select Install Now to install Python with default settings, or choose
Customize to enable or disable features.
Install Now
C:¥Users¥zaurus¥AppData¥Local¥Programs¥Python¥Python37
Includes IDLE, pip and documentation
Creates shortcuts and file associations
Customize installation
Choose location and features
python
for
windows
Install launcher for all users (recommended)
Add Python 3.7 to PATH
Cancel


Python 3.7.8 (64-bit) Setup
Optional Features
Documentation
Installs the Python documentation file.
pip
Installs pip, which can download and install other Python packages.
tcl/tk and IDLE
Installs tkinter and the IDLE development environment.
Python test suite
Installs the standard library test suite.
py launcher
for all users (requires elevation)
Installs the global 'py' launcher to make it easier to start Python.
python
for
windows
Back
Next
Cancel


Python 3.7.8 (64-bit) Setup
Advanced Options
Install for all users
Associate files with Python (requires the py launcher)
Create shortcuts for installed applications
Add Python to environment variables
Precompile standard library
Download debugging symbols
Download debug binaries (requires VS 2015 or later)
Customize install location
C:\Python37
Browse
Back
Install
Cancel
python
for
windows


Python 3.7.8 (64-bit) Setup
Setup was successful
Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.
New to Python? Start with the online tutorial and documentation.
See what's new in this release.
Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.
Close
python
for
windows

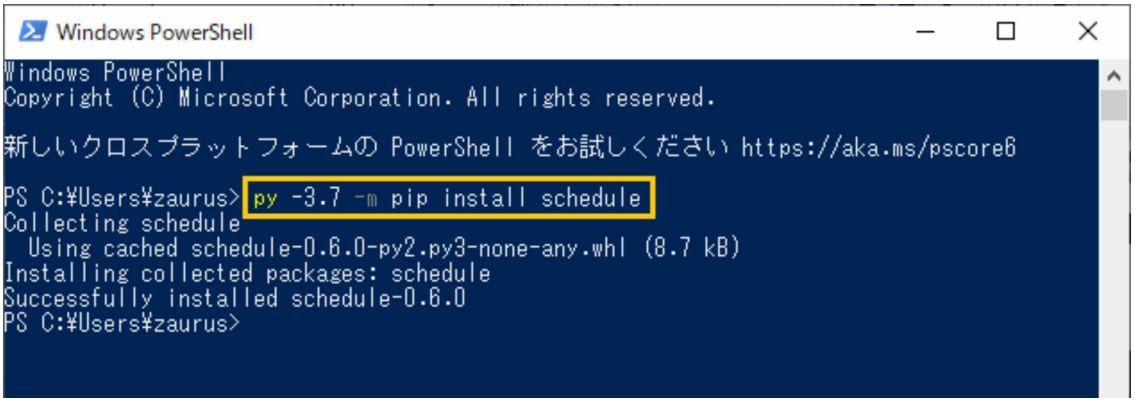
3　schedule パッケージを追加

時刻表どおりに列車を発車させるため、Python3.7 に schedule パッケージを追加します。

Windows のスタートボタンを右クリックして「 Windows PowerShell 」を起動します。

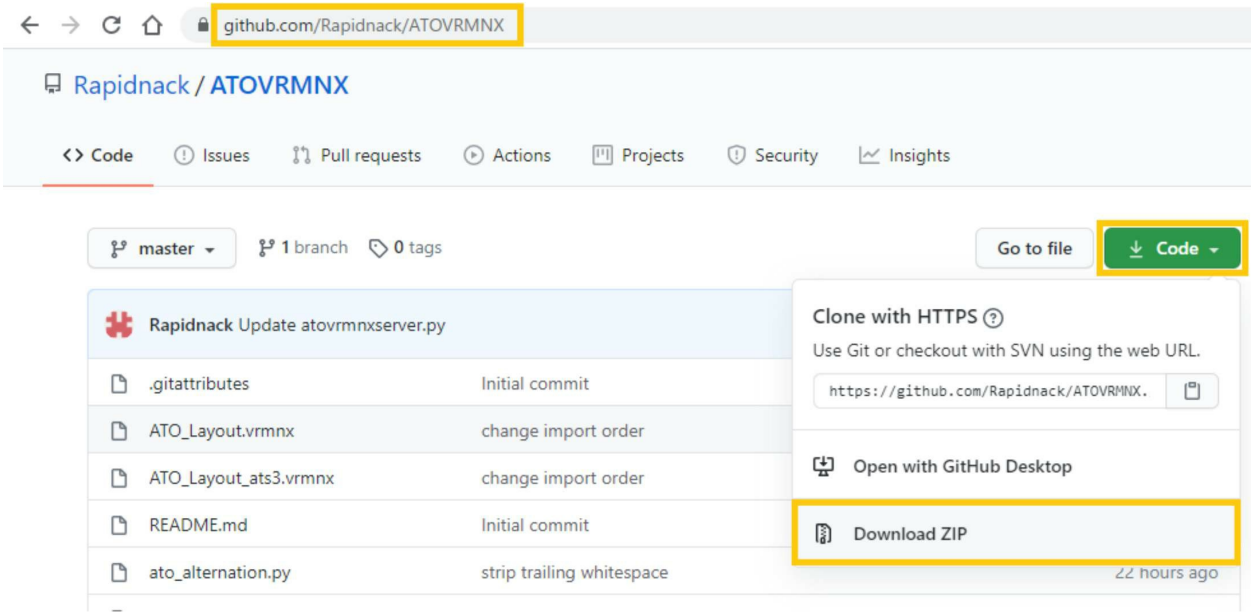
pip コマンドで schedule パッケージをインストールします。

```
py -3.7 -m pip install schedule
```

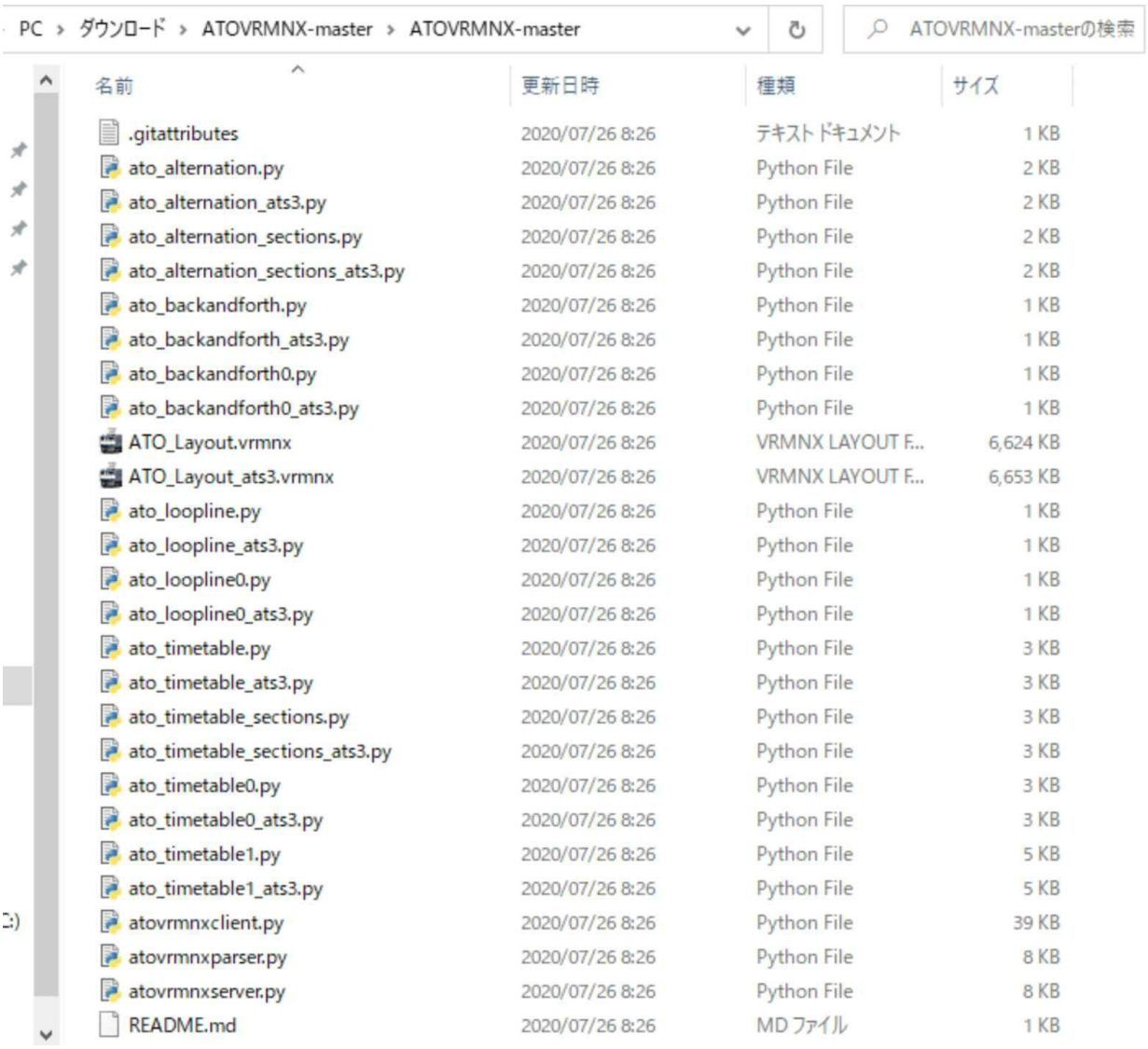


4　　レイアウトファイル、モジュール、スクリプトを取得

https://github.com/Rapidnack/ATOVRMNXからファイル一式纏めてダウンロードします。



ダウンロードして解凍すると次のようなフォルダ構成になります。


PC › ダウンロード › ATOVRMNX-master › ATOVRMNX-master
ATOVRMNX-masterの検索
名前
更新日時
種類
サイズ
.gitattributes 2020/07/26 8:26 テキスト ドキュメント 1 KB
ato_alternation.py 2020/07/26 8:26 Python File 2 KB
ato_alternation_ats3.py 2020/07/26 8:26 Python File 2 KB
ato_alternation_sections.py 2020/07/26 8:26 Python File 2 KB
ato_alternation_sections_ats3.py 2020/07/26 8:26 Python File 2 KB
ato_backandforth.py 2020/07/26 8:26 Python File 1 KB
ato_backandforth_ats3.py 2020/07/26 8:26 Python File 1 KB
ato_backandforth0.py 2020/07/26 8:26 Python File 1 KB
ato_backandforth0_ats3.py 2020/07/26 8:26 Python File 1 KB
ATO_Layout.vrmnx 2020/07/26 8:26 VRMNX LAYOUT F... 6,624 KB
ATO_Layout_ats3.vrmnx 2020/07/26 8:26 VRMNX LAYOUT F... 6,653 KB
ato_loopline.py 2020/07/26 8:26 Python File 1 KB
ato_loopline_ats3.py 2020/07/26 8:26 Python File 1 KB
ato_loopline0.py 2020/07/26 8:26 Python File 1 KB
ato_loopline0_ats3.py 2020/07/26 8:26 Python File 1 KB
ato_timetable.py 2020/07/26 8:26 Python File 3 KB
ato_timetable_ats3.py 2020/07/26 8:26 Python File 3 KB
ato_timetable_sections.py 2020/07/26 8:26 Python File 3 KB
ato_timetable_sections_ats3.py 2020/07/26 8:26 Python File 3 KB
ato_timetable0.py 2020/07/26 8:26 Python File 3 KB
ato_timetable0_ats3.py 2020/07/26 8:26 Python File 3 KB
ato_timetable1.py 2020/07/26 8:26 Python File 5 KB
ato_timetable1_ats3.py 2020/07/26 8:26 Python File 5 KB
atovrmnxclient.py 2020/07/26 8:26 Python File 39 KB
atovrmnxparser.py 2020/07/26 8:26 Python File 8 KB
atovrmnxserver.py 2020/07/26 8:26 Python File 8 KB
README.md 2020/07/26 8:26 MD ファイル 1 KB

5　リモート制御にした理由

VRMNX は鉄道模型シミュレーターですからメインタスクは画面の表示更新です。そのため現在の VRMNX のスクリプトでは frame イベントごとに細切れで処理を進める必要があります。

C# でウィンドウにボタンとテキストボックスが載っているだけのアプリケーションを作る場合を考えると、画面のボタンをクリックしたときのイベントハンドラには一塊の処理を記述できます。

```
private void button1_Click(object sender, EventArgs e)
{
        ファイルからデータを読み込む。
        計算する。
        結果を画面に表示する。
}
```

frame イベントごとに細切れで処理を進める方式は慣れていないとなかなかハードです。

frame イベントを気にしなくて良いように、別アプリケーションから VRMNX にコマンドを送って API を実行できれば、別アプリケーション側で VRMATS の catch イベントに対して一塊の処理を記述できます。

```
def sequence(train):
   train.stop(550)   # 550mm 減速して停止するまで待つ
    time.sleep(3)          #  停車時間 3 秒
    train.start(400)  #  走行速度まで 400mm で加速
```

当然 catch イベント処理中も別の catch イベントが発生するので、それぞれ別スレッドで実行する必要があります。

6　TCP サーバーの起動

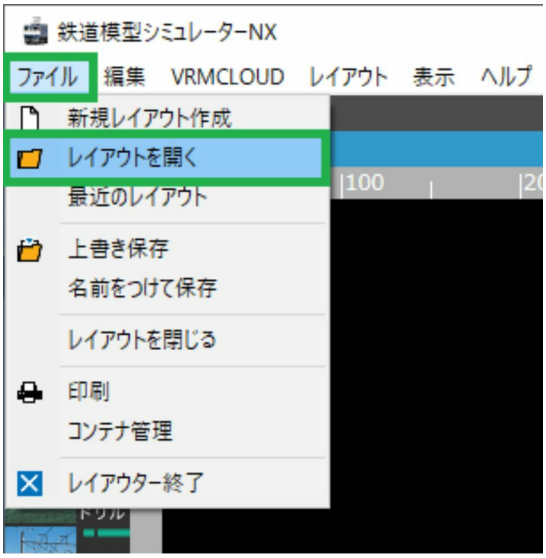
GitHub よりダウンロードしたフォルダには TCP サーバー機能に使用する2つのモジュールが入っています。

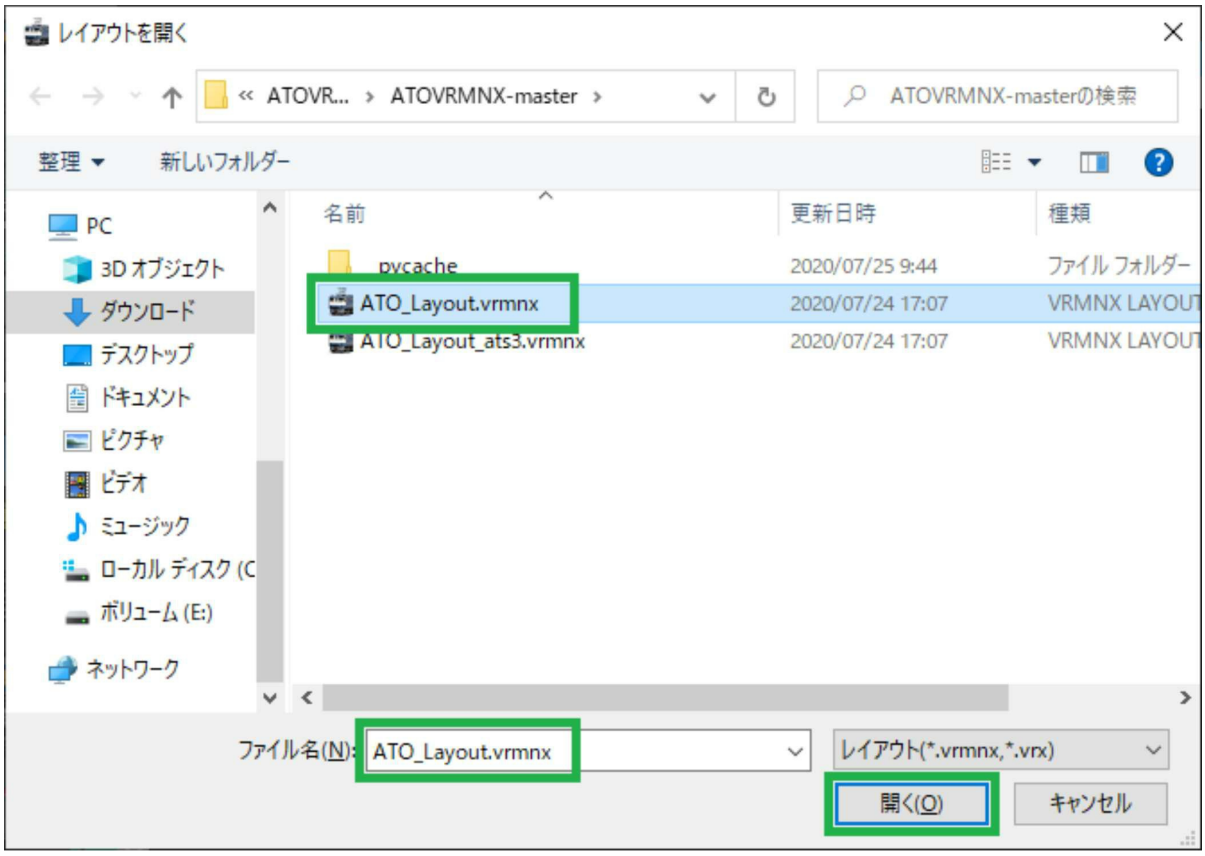
atovrmnxserver.py atovrmnxparser.py

同じフォルダに入っている2つのレイアウトファイルのレイアウトスクリプトには、既に TCP サーバー機能に必要なコードを追加してあります。

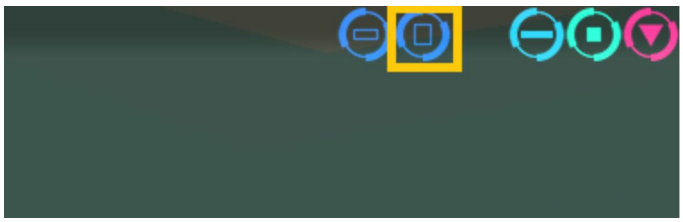
ATO_Layout.vrmnx ATO_Layout_ats3.vrmnx

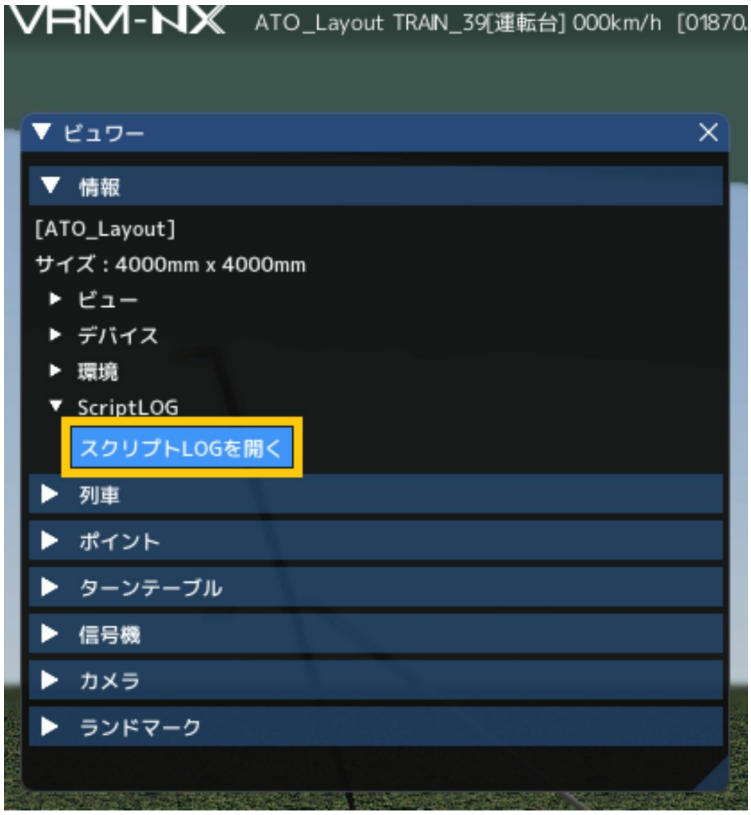
「 ATO_Layout.vrmnx 」を開いて「運転」または「試運転」をクリックしてビュワーを起動します。

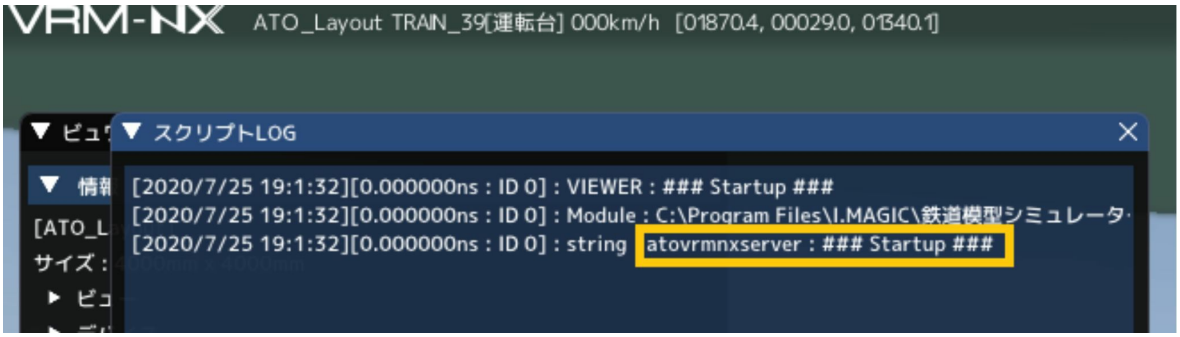




ビュワー起動時に TCP サーバーも起動することを「スクリプト LOG 」画面で確認できます。







GitHub よりダウンロードしたレイアウトファイル以外で TCP サーバーを起動するには3つの手順が必要です。

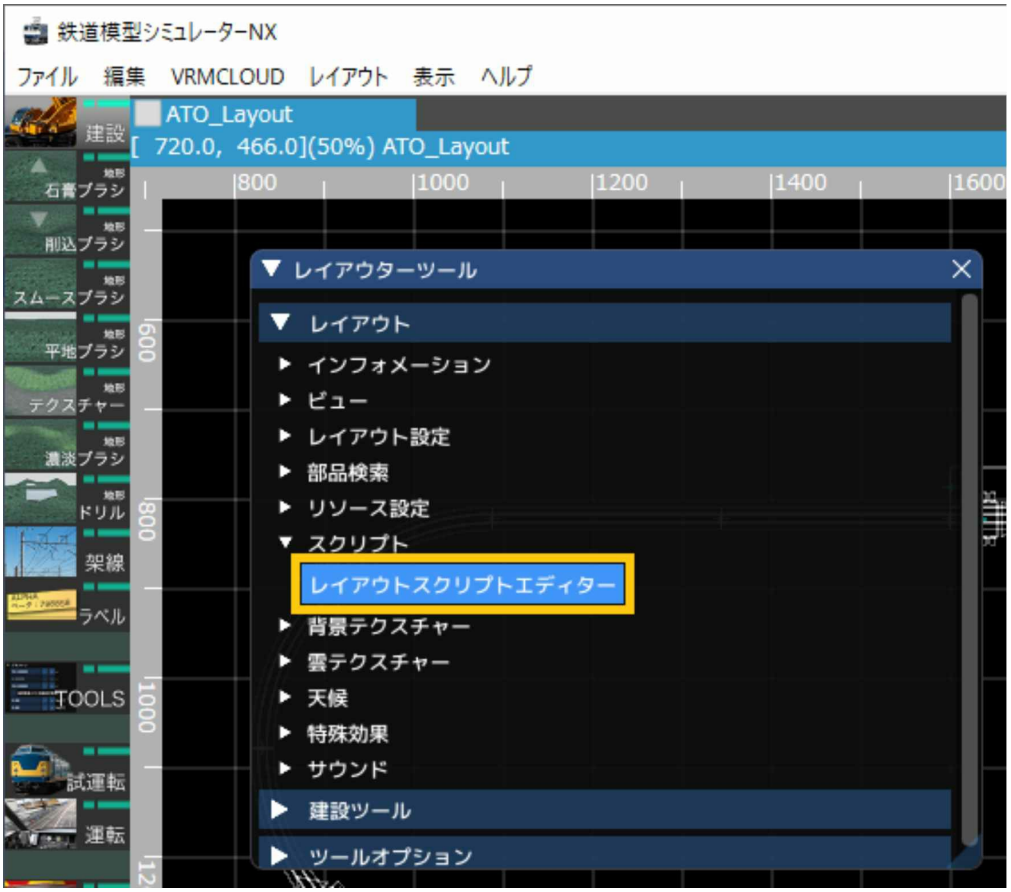
1. **Python3.7** のインストール場所に合わせて **atovrmnxparser** モジュール内のモジュール検索パスを変更します。

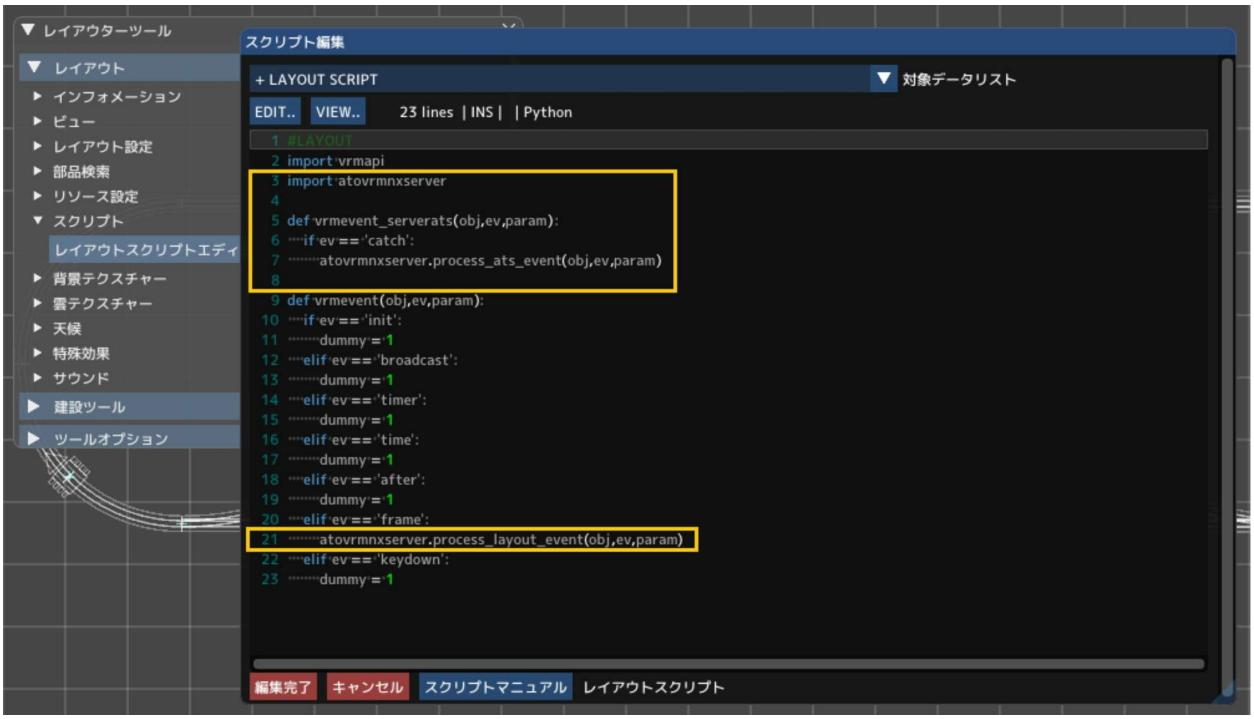
```
sys.path.append("C:¥Python37¥Lib")
sys.path.append("C:¥Python37¥DLLs")
sys.path.append("C:¥Python37¥Lib¥site-packages")
```

2. 次の2つのモジュールをレイアウトファイルと同じフォルダに配置します。

atovrmnxserver.py
atovrmnxparser.py

3. レイアウトのスクリプトに、**atovrmnxparser** モジュールを呼び出すコードを追加します。






```
#LAYOUT
import vrmapi
import atovrmnxserver

def vrmevent_serverats(obj,ev,param):
    if ev == 'catch':
        atovrmnxserver.process_ats_event(obj,ev,param)

def vrmevent(obj,ev,param):
    if ev == 'init':
        dummy = 1
    elif ev == 'broadcast':
        dummy = 1
    elif ev == 'timer':
        dummy = 1
    elif ev == 'time':
        dummy = 1
    elif ev == 'after':
        dummy = 1
```

```
    elif ev == 'frame':
        atovrmnxserver.process_layout_event(obj,ev,param)
    elif ev == 'keydown':
        dummy = 1
```

ビュワーの表示中、TCP サーバーはクライアントからの接続要求を受け付けます。
コマンド用 TCP ポート番号 54001
イベント用 TCP ポート番号 54002

接続後、クライアントから送られてくる '¥n' で終端したコマンド文字列を受信すると、atovrmnxparser モジュールが解釈して vrmapi モジュールの API を実行します。
例：
'SetVoltage(0.5)¥n'　　アクティブ列車対象
'Turn()¥n'　　アクティブ列車対象
'LAYOUT().GetTrain(39).AutoSpeedCTRL(400, 0.5)¥n'
'LAYOUT().GetPoint(73).SetBranch(1)¥n'
実行できる API は、atovrmnxparser モジュールに処理を記述してあるものだけです。他の API もクライアントから実行したい場合は atovrmnxparser モジュールに処理を追加してください。

catch イベントは '¥n' で終端した文字列に変換されてクライアントに送信されます。
例：
'catch 79 40 1 1¥n'　　内容：VRMATS[79]、VRMTrain[40]、順方向、先頭車輪

7　統合開発環境「 IDLE 」で列車を操作

クライアント側は別アプリケーションなので開発言語を自由に選択できますが、VRMNX のスクリプトが Python なのでクライアント側も Python を選択しました。

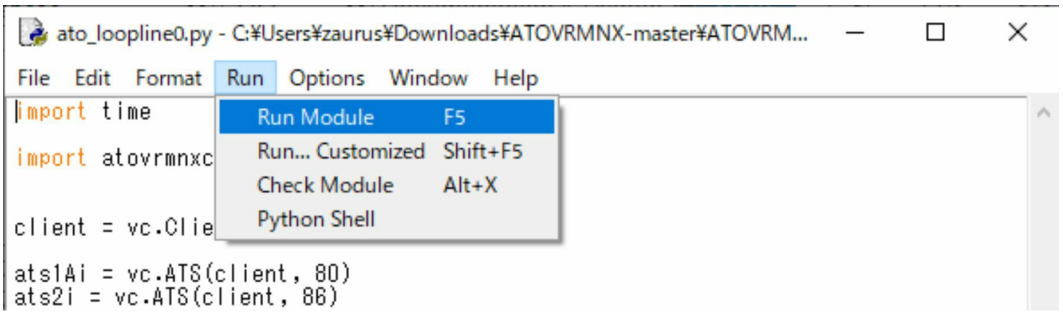
GitHub からダウンロードしたフォルダに入っている atovrmnxclient モジュールをインポートすると、リモート自動運転用のスクリプトを簡単に作ることができます。

atovrmnxclient.py

スクリプトの作成には Python の統合開発環境「 IDLE 」が便利です。

「 IDLE 」はインタラクティブモードの Shell 画面とファイルを編集して実行する Editor 画面に分けて説明されることが多いですが、組み合わせて使うことをお勧めします。

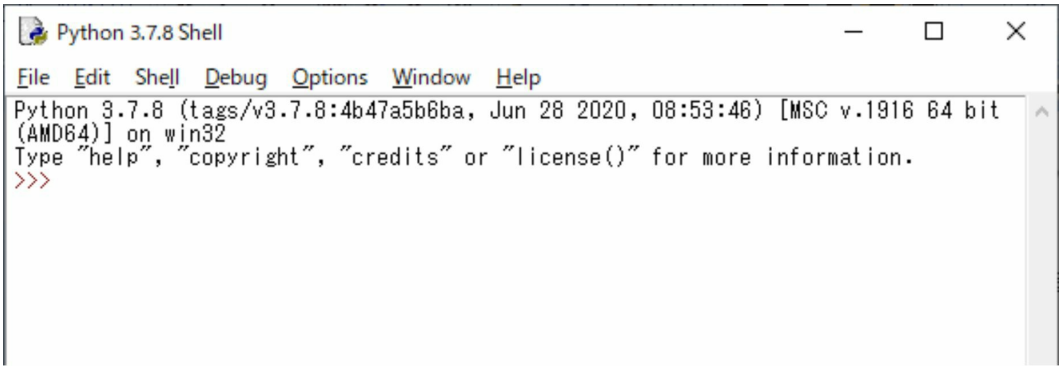
Editor 画面の「 Run Module F5 」メニューをクリックすると、



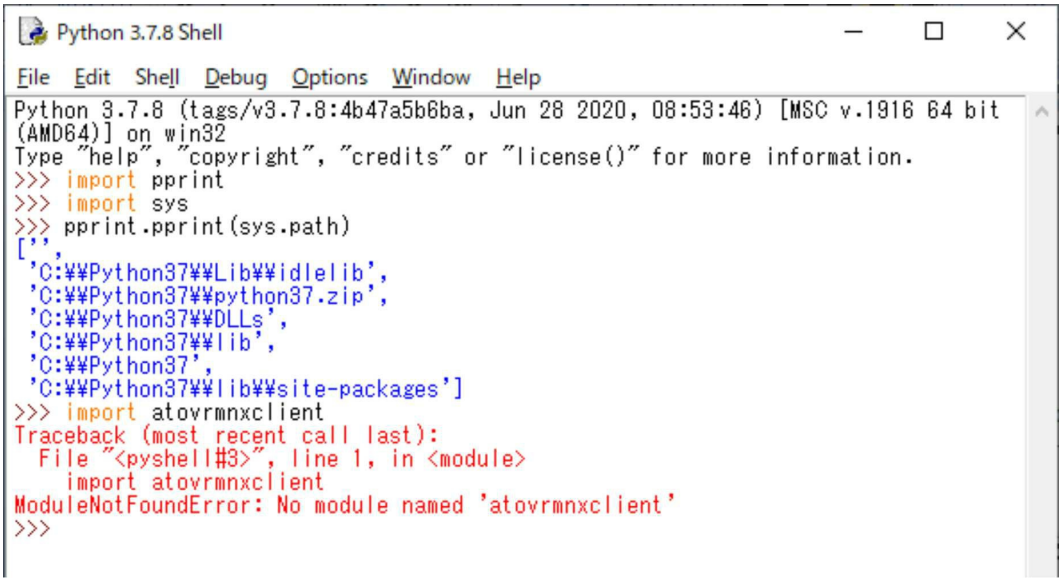
Python インタープリタを一度初期化してから Editor 画面が保存したファイルの内容を実行し、**Python インタープリタの状態を維持したまま Shell 画面でインタラクティブモードを開始します。**インタラクティブモードで必要になる手順を Editor 画面に書いておいて「 Run Module F5 」をクリックすればインタラクティブモードの準備完了、という使い方ができます。

「 IDLE 」を起動してみましょう。

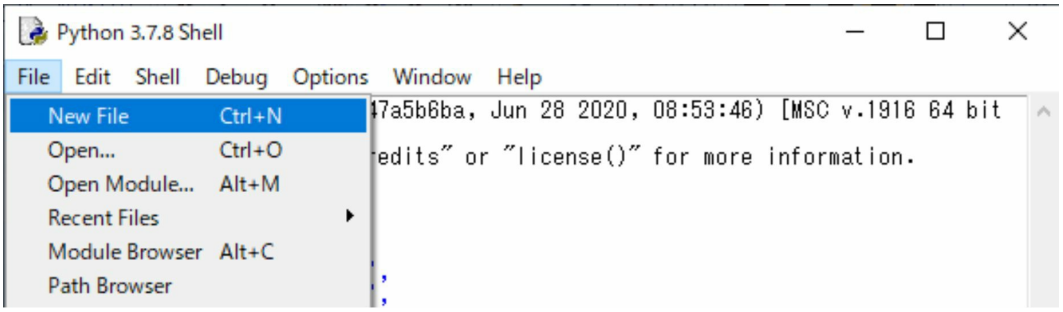
Shell 画面が表示されます。



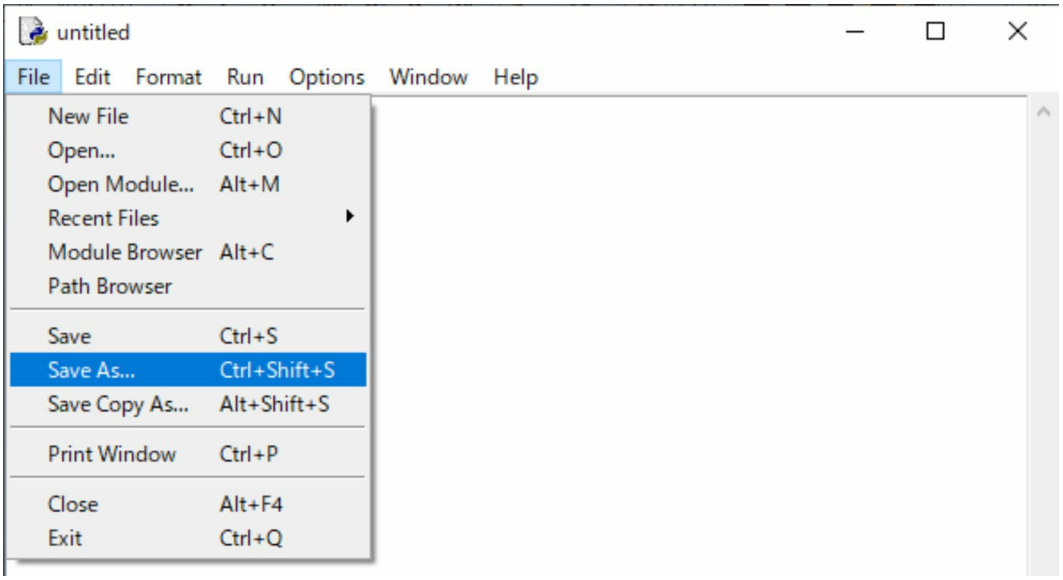
モジュール検索パスを確認すると当然 Python の標準パスだけで、GitHub からダウンロードしたフォルダに入っている atovrmnxclient モジュールをインポートできません。

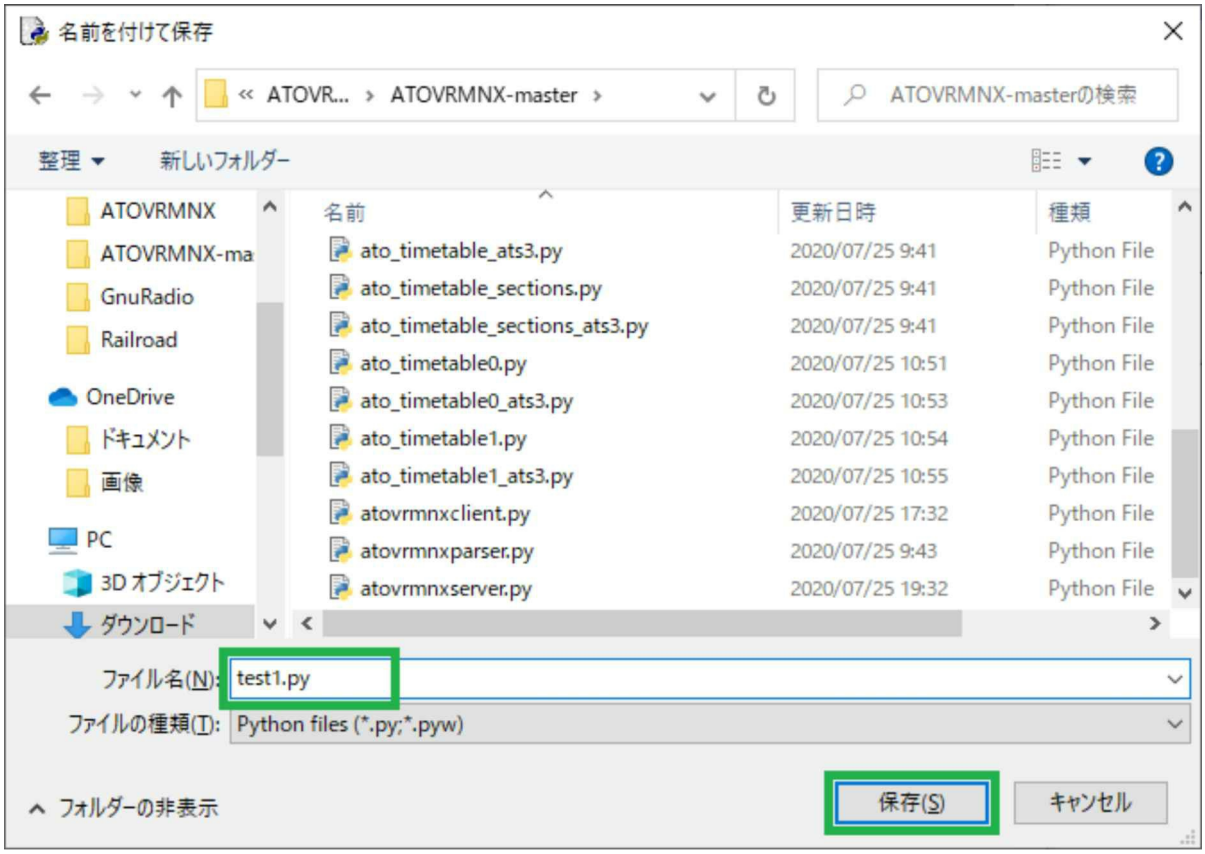


新しい Editor 画面を表示します。



表示された Editor 画面を atovrmnxclient モジュールと同じフォルダに適当な名前で保存します。





Editor 画面で「 Run Module F5 」をクリックしてから、先ほどと同じコマンドを

入力します。ALT+p で入力履歴を戻れます。ALT+n で進めます。モジュール検索パスに Editor 画面の保存先が追加されているので同じフォルダに在る atovrmnxclient モジュールをインポートできました。

```
>>>
== RESTART: C:/Users/zaurus/Downloads/ATOVRMNX-master/ATOVRMNX-master/test1.py =
>>> import pprint
>>> import sys
>>> pprint.pprint(sys.path)
['C:/Users/zaurus/Downloads/ATOVRMNX-master/ATOVRMNX-master',
 'C:¥¥Python37¥¥Lib¥¥idlelib',
 'C:¥¥Python37¥¥python37.zip',
 'C:¥¥Python37¥¥DLLs',
 'C:¥¥Python37¥¥lib',
 'C:¥¥Python37',
 'C:¥¥Python37¥¥lib¥¥site-packages']
>>> import atovrmnxclient
>>>
```

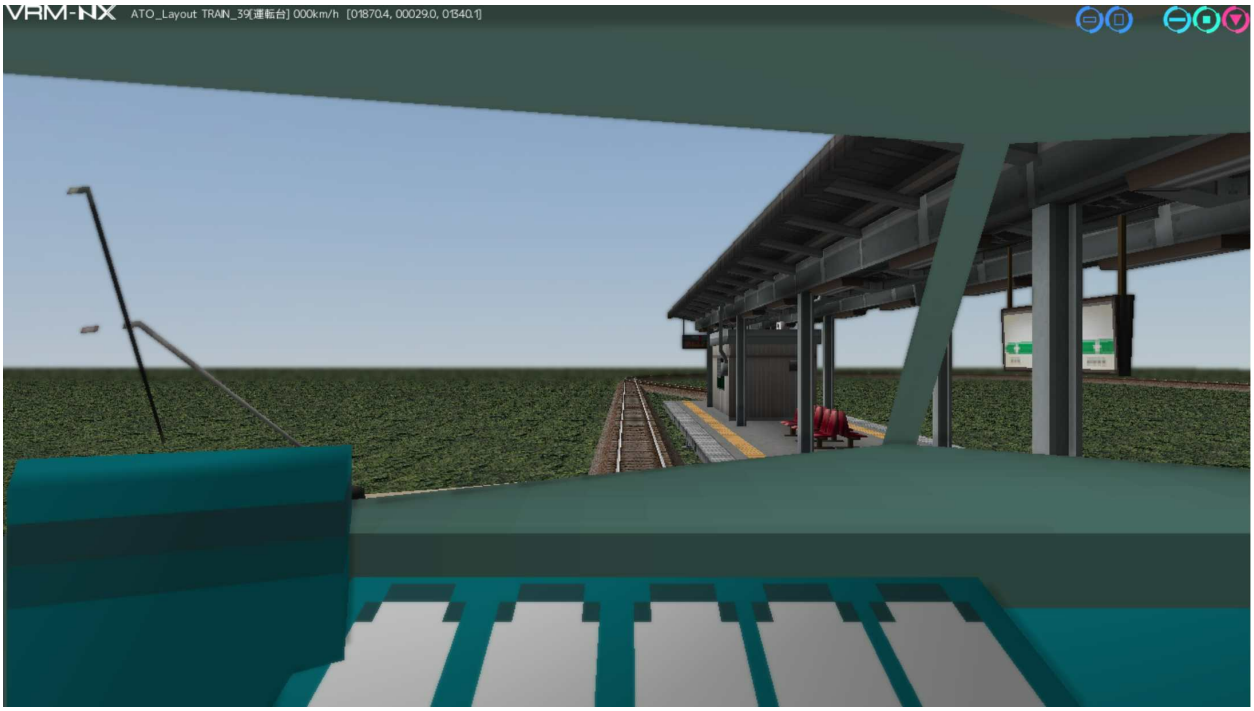
再度 Editor 画面で「 Run Module F5 」をクリックしてから、Python の「名前空間」を globals() で確認してみます。アンダースコア2つで始まる名前を無視すると、インポートした「 pprint 」だけが追加されています。

この状態では __builtins__ に含まれる print() 、globals() 等の組み込みオブジェクトと「 pprint 」だけが使用可能です。

```
>>>
== RESTART: C:/Users/zaurus/Downloads/ATOVRMNX-master/ATOVRMNX-master/test1.py =
>>> import pprint
>>> pprint.pprint(globals())
{'__annotations__': {},
 '__builtins__': <module 'builtins' (built-in)>,
 '__doc__': None,
 '__file__': 'C:/Users/zaurus/Downloads/ATOVRMNX-master/ATOVRMNX-master/test1.py
',
 '__loader__': <class '_frozen_importlib.BuiltinImporter'>,
 '__name__': '__main__',
 '__package__': None,
 '__spec__': None,
 'pprint': <module 'pprint' from 'C:¥¥Python37¥¥lib¥¥pprint.py'>}
>>>
```

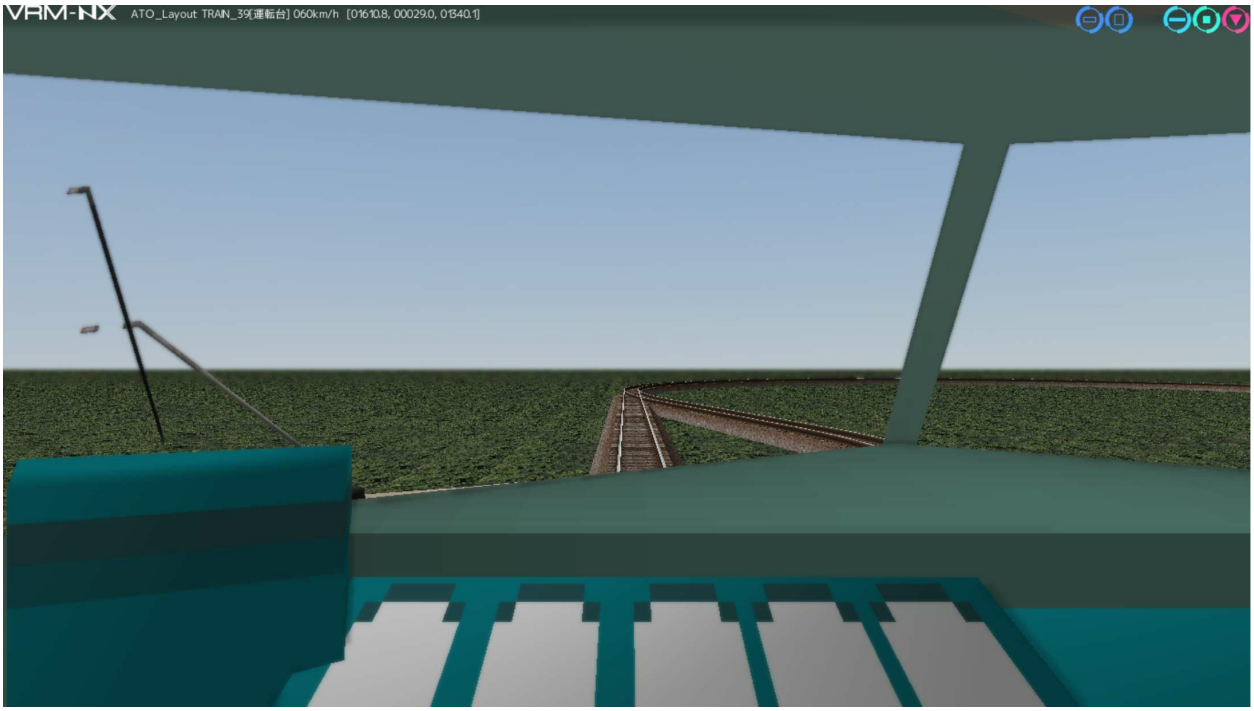
では、再度 atovrmnxclient モジュールをインポートして TCP サーバーと接続しましょう。

VRMNX を起動し、GitHub からダウンロードしたフォルダに入っている「 ATO_Layout.vrmnx 」を開いて「運転」をクリックしてビュワーを起動します。



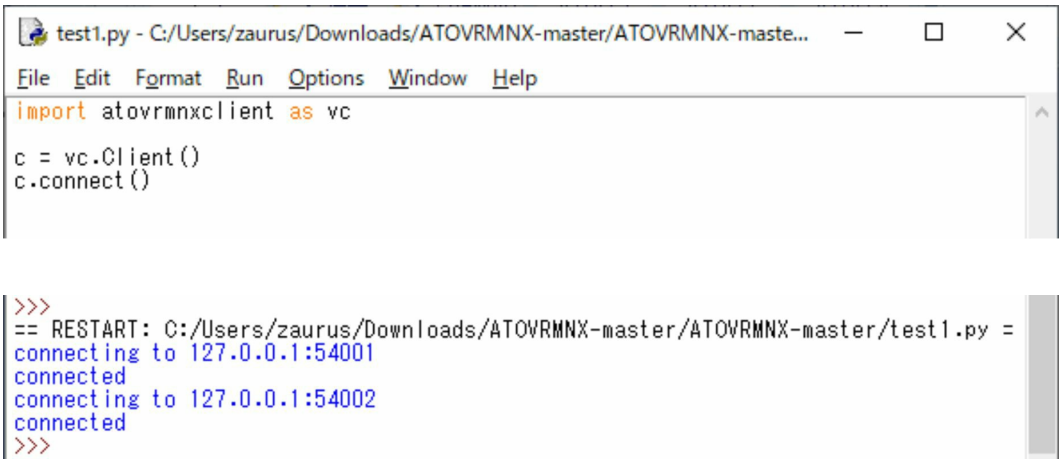
atovrmnxclient は長いので「 as vc 」を付けて「 vc 」という別名でインポートします。Client オブジェクトを生成して TCP サーバーと接続、文字列 'SetVoltage(0.5)' を TCP サーバーに送ってアクティブな列車を発車させます。

```
>>>
>>> import atovrmnxclient as vc
>>> c = vc.Client()
>>> c.connect()
connecting to 127.0.0.1:54001
connected
connecting to 127.0.0.1:54002
connected
<Thread(Thread-1, started daemon 13336)>
>>> c.send('SetVoltage(0.5)')
>>>
```



一度ビュワーを終了し再度「運転」をクリックします。

Editor 画面に接続までの手順を入力して保存し「 Run Module F5 」をクリックすると、Shell 画面は TCP サーバーと接続した状態でインタラクティブモードを開始します。



Python の「名前空間」を globals() で確認してみると、Shell 画面でインポートした「 pprint 」の他に、Editor 画面に記述した atovrmnxclient モジュールに付けた別名「 vc 」、Client オブジェクト「 c 」も追加されています。

```
>>>
== RESTART: C:¥Users¥zaurus¥Downloads¥ATOVRMNX-master¥ATOVRMNX-master¥test1.py =
connecting to 127.0.0.1:54001
connected
connecting to 127.0.0.1:54002
connected
>>> import pprint
>>> pprint.pprint(globals())
{'__annotations__': {},
 '__builtins__': <module 'builtins' (built-in)>,
 '__doc__': None,
 '__file__': 'C:¥¥Users¥¥zaurus¥¥Downloads¥¥ATOVRMNX-master¥¥ATOVRMNX-master¥¥te
st1.py',
 '__loader__': <class '_frozen_importlib.BuiltinImporter'>,
 '__name__': '__main__',
 '__package__': None,
 '__spec__': None,
 'c': <atovrmnxclient.Client object at 0x00000018FDE7CFB08>,
 'pprint': <module 'pprint' from 'C:¥¥Python37¥¥lib¥¥pprint.py'>,
 'vc': <module 'atovrmnxclient' from 'C:¥¥Users¥¥zaurus¥¥Downloads¥¥ATOVRMNX-mas
ter¥¥ATOVRMNX-master¥¥atovrmnxclient.py'>}
>>>
```

「 c.send('SetVoltage(0.5)') 」と入力するだけで列車が発車します。

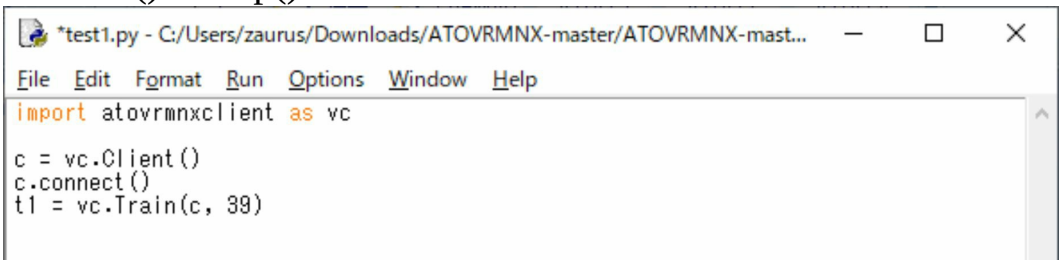
```
>>>
>>> c.send('SetVoltage(0.5)')
>>>
```

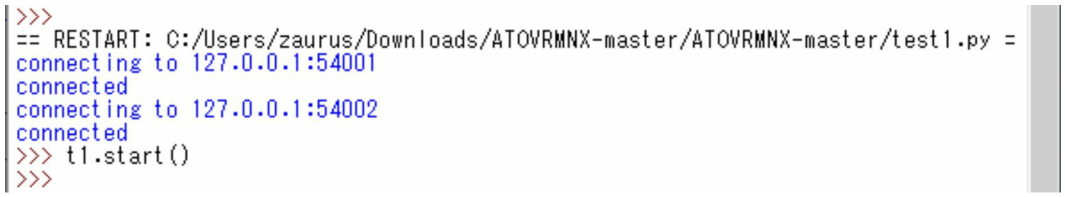
アクティブな列車に対して次の API を文字列として Client オブジェクトの send() メソッドで送信できます。

AutoSpeedCTRL()
GetDirection()
GetVoltage()
GetID()
SetTimerVoltage()
SetVoltage()
Turn()

アクティブな列車用の API は動作確認専用で自動運転では使用しません。

自動運転では列車を ID で指定した Train オブジェクトを生成して Train オブジェクトの start() 、stop() メソッド等を使います。





これより自動運転で使用する Client クラス、Train クラス、Point クラス、ATS クラスを順に説明していきます。

8 Client クラス

Client クラスは VRMNX 用に作成したサーバーモジュール atovrmnxserver に接続するクライアントのクラスです。

8.1　コンストラクタ

def __init__(self):

8.2 メソッド

def connect(self, address='127.0.0.1', commandport=54001, eventport=54002):

コマンド用とイベント用の2つの TCP ソケットをサーバーに接続する。

接続後イベント受信スレッドを開始する。

Args:
address (str): サーバーアドレス。
commandport (int): コマンド用 TCP ソケットのポート番号。
eventport (int): イベント用 TCP ソケットのポート番号。

Returns:
threading.Thread: イベント受信スレッド。

def disconnect(self):

サーバーとの接続を切る。

def send(self, command):

コマンド文字列をサーバーに送信する。

Args:
command (str): '¥n' で終端されたコマンド文字列。

def sendquery(self, command):

コマンド文字列をサーバーに送信し、受信した応答文字列を返す。

Args:
command (str): '¥n' で終端されたコマンド文字列。

Returns:
str: 受信した応答文字列（ '¥n' 削除済み）。

def startthread(self, sequence, args=None):

関数を新しいスレッドで開始する。

Args:
sequence (str): スレッドで実行される関数。
args (obj): 関数に渡される引数。タプルの場合はタプルの中身が複数の引数になる。
例：
startthread(func, arg) => func(arg)
startthread(func, (arg1, arg2, arg3)) => func(arg1, arg2, arg3)

9 Train クラス

Train クラスは VRMNX の VRMTrain に対応するクラスです。

9.1 コンストラクタ

def __init__(self, client, id, code=None, number=None, startdistance=200, stopdistance=50, voltage=0.5):

Args:
client (Client): クライアントオブジェクト。
id (int): VRMTrain の ID 。
code (int): VRMTrain の種別コード。
number (str): VRMTrain の列車番号。
startdistance (float): start() のデフォルト加速距離 mm 。
stopdistance (float): stop() のデフォルト減速距離 mm 。
voltage (float): start() のデフォルト走行速度の電圧。

9.2　メソッド

def send(self, command):

'LAYOUT().GetTrain(id).' + コマンド文字列をサーバーに送信する。

Args:
  command (str): '¥n' で終端されたコマンド文字列。

def sendquery(self, command):

'LAYOUT().GetTrain(id).' + コマンド文字列をサーバーに送信し、受信した応答文字列を返す。

Args:
  command (str): '¥n' で終端されたコマンド文字列。

Returns:
  str: 受信した応答文字列（ '¥n' 削除済み）。

def AutoSpeedCTRL(self, distance, voltage):

vrmapi の同名 API を実行。

def GetDirection(self):

vrmapi の同名 API を実行。

def GetVoltage(self):

vrmapi の同名 API を実行。

def SetTimerVoltage(self, sec, voltage):

vrmapi の同名 API を実行。

def SetTrainCode(self, code=None):

vrmapi の同名 API を実行。

code を省略すると code プロパティが使用される。
code プロパティも設定されていないときは何もしない。

def SetTrainNumber(self, number=None):

vrmapi の同名 API を実行。

number を省略すると number プロパティが使用される。
number プロパティも設定されていないときは何もしない。

def SetVoltage(self, voltage):

vrmapi の同名 API を実行。

def Turn(self):

vrmapi の同名 API を実行。

def start(self, distance=None, voltage=None):

AutoSpeedCTRL(distance, voltage) を実行。

distance を省略すると startdistance プロパティが使用される。
voltage を省略すると voltage プロパティが使用される。

Args:
distance (float): 加速距離 mm 。
voltage (float): 走行速度の電圧。

def stop(self, distance=None, wait=True):

AutoSpeedCTRL(distance, 0.0) を実行し、列車が停止するまで待つ。

distance を省略すると stopdistance プロパティが使用される。

Args:
distance (float): 減速距離 mm 。
wait (bool): True なら列車が停止するまで待つ。

def waituntilstop(self):

列車が停止するまで待つ。

9.3　プロパティ

code

VRMTrain の種別コードを取得・設定する。

number

VRMTrain の列車番号の文字列を取得・設定する。

startdistance

start() のデフォルト加速距離 mm を取得・設定する。

stopdistance

stop() のデフォルト減速距離 mm を取得・設定する。

voltage

start() のデフォルト走行速度の電圧を取得・設定する。

走行中なら列車の電圧も即時変更する。

10　Point クラス

Point クラスは VRMNX の VRMPoint に対応するクラスです。

10.1　コンストラクタ

def __init__(self, client, id):

Args:
client (Client): クライアントオブジェクト。
id (int): VRMPoint の ID 。

10.2　メソッド

def send(self, command):

'LAYOUT().GetPoint(id).' + コマンド文字列をサーバーに送信する。

Args:
  command (str): '¥n' で終端されたコマンド文字列。

def sendquery(self, command):

'LAYOUT().GetPoint(id).' + コマンド文字列をサーバーに送信し、受信した応答文字列を返す。

Args:
  command (str): '¥n' で終端されたコマンド文字列。

Returns:
  str: 受信した応答文字列（ '¥n' 削除済み）。

def GetBranch(self):

vrmapi の同名 API を実行。

def SetBranch(self, branch):

vrmapi の同名 API を実行。

def SwitchBranch(self):

vrmapi の同名 API を実行。

11　ATS クラス

ATS クラスは VRMNX の VRMATS に対応するクラスです。

11.1　コンストラクタ

def __init__(self, client, id):

Args:
client (Client): クライアントオブジェクト。
id (int): VRMATS の ID 。

11.2　メソッド

def send(self, command):

'LAYOUT().GetATS(id).' + コマンド文字列をサーバーに送信する。

Args:
command (str): '¥n' で終端されたコマンド文字列。

def SetUserEventFunction(self, funcname):

vrmapi の同名 API を実行。

def ClearUserEventFunction(self):

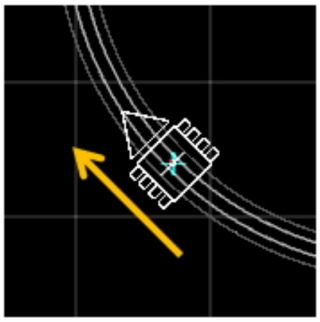
vrmapi の同名 API を実行。

11.3　プロパティ

forward

VRMATS オブジェクトの順方向の列車検出時に実行されるユーザー定義関数を取得・設定する。

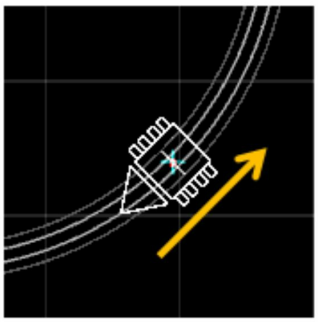
引数を渡すときは (ユーザー定義関数 , arg1, arg2, ...) のようなタプルを設定する。



reverse

VRMATS オブジェクトの逆方向の列車検出時に実行されるユーザー定義関数を取得・設定する。

引数を渡すときは (ユーザー定義関数 , arg1, arg2, ...) のようなタプルを設定する。



12　3 連 ATS

メーカーが推奨している動作環境より低いスペックの PC で動作させている方向けの情報です。

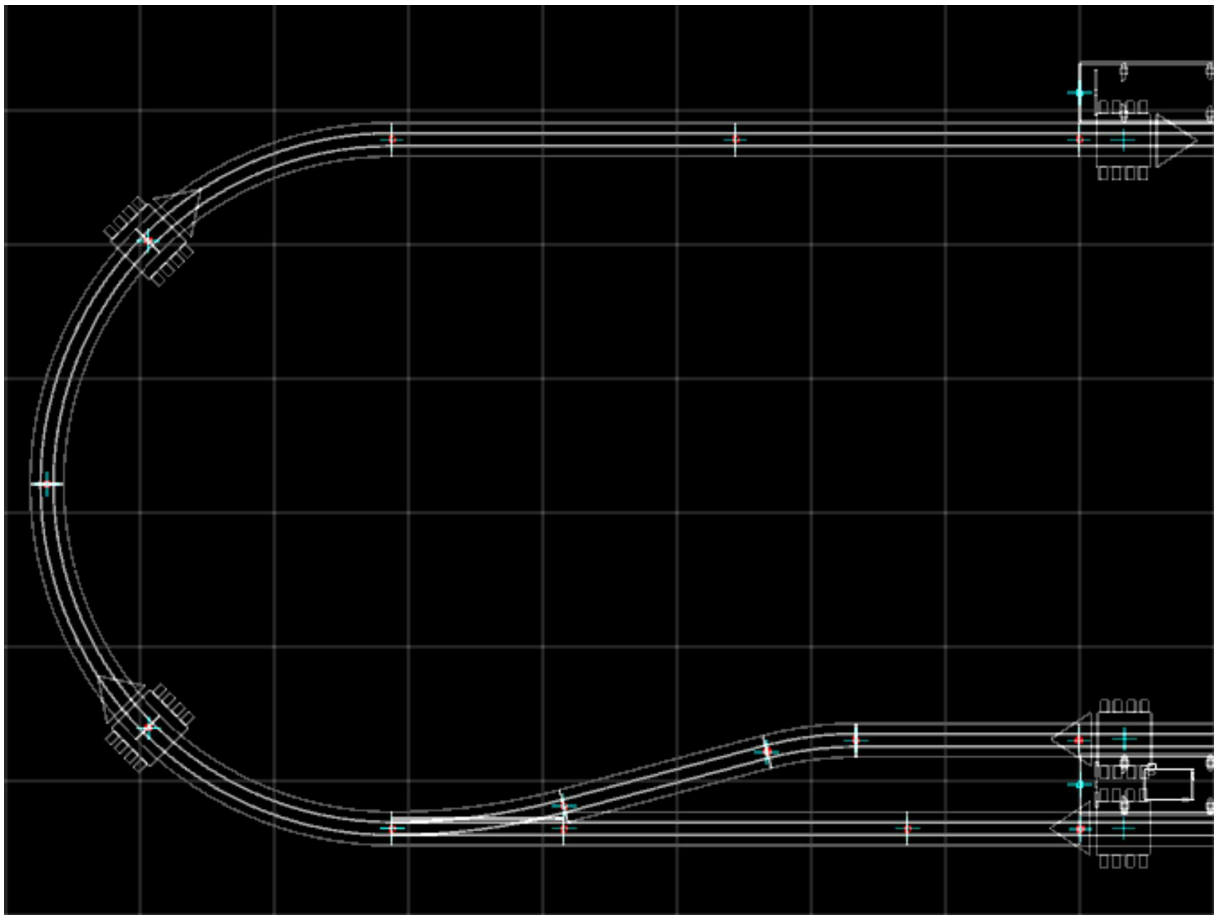
VRMNX のリモート自動運転を Pentium G3220 の PC で連続動作させていると、いつの間にかポイント切り替えミス等で列車が衝突しています。

試しに 1 か所の複数の ATS を配置して最初に発生した catch イベントにだけ対応する機能を追加したところ、問題なく連続動作できるようになりました。

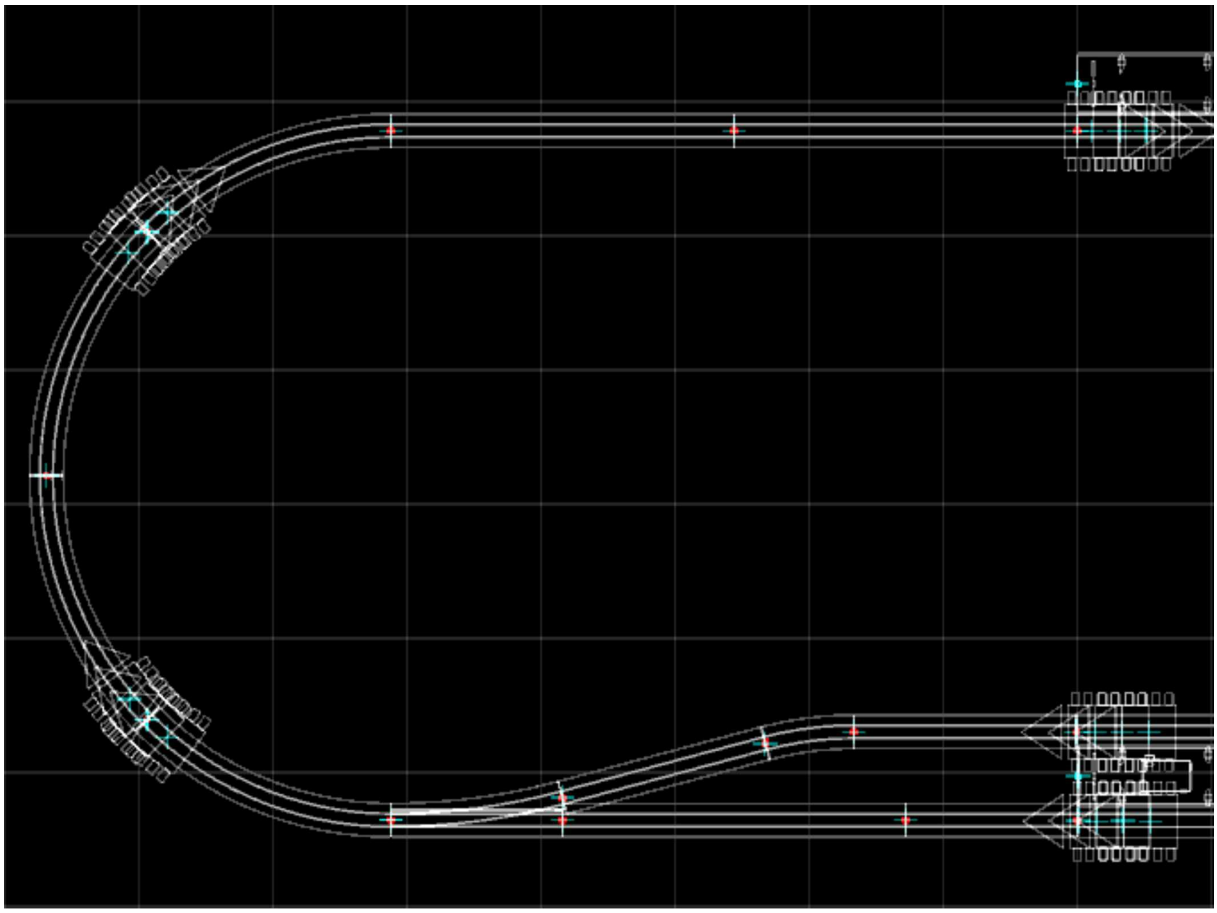
下記の黄色の行は ID=79 、89 、99 の順に3つ ATS を並べた箇所で、先頭の ID=79 の ATS が列車を見逃して次の ID=89 の ATS が検出したときの IDLE の Shell 画面の表示です。

```
catch 81 39 1 1 sequence1Ao
catch 86 40 1 1
catch 84 39 1 1
catch 85 39 1 1
catch 87 40 1 1
catch 88 40 1 1
(79,) catch 89 40 1 1 sequence1ii
catch 86 39 1 1
catch 82 40 1 1
catch 87 39 1 1
catch 88 39 1 1
catch 79 39 1 1 sequence1ii
```

ダウンロードしたフォルダにあるレイアウト「 ATO_Layout_ats3.vrmnx 」と「 ato_loopline0_ats3.py 」等の「 _ats3 」の付いたスクリプトを組み合わせて、列車の見逃しが起きていないか試してみてください。



ATO_Layout.vrmnx



ATO_Layout_ats3.vrmnx

「 ato_loopline0.py 」の ATS オブジェクト生成。

```
ats1Ai = vc.ATS(client, 80)
ats2i = vc.ATS(client, 86)
```

「 ato_loopline0_ats3.py 」の ATS オブジェクト生成。

```
ats1Ai = vc.ATS(client, (80, 90, 100))
ats2i = vc.ATS(client, (86, 96, 106))
```

13　サンプルスクリプトの基本構成

サンプルスクリプトはほぼ次のような構成になっています。

```
パッケージをインポート

Client オブジェクト生成
ATS オブジェクト生成
Train オブジェクト生成

catch イベント用関数定義
ATS オブジェクトに catch イベント用関数登録

メイン関数定義
メイン関数実行
```

例として GitHub からダウンロードしたフォルダに入っている「 ato_loopline0.py 」を見てみます。

ato_loopline0 スクリプトは、1つの列車が2つの駅に停車しつつ周回します。駅の入口に配置した ATS の catch イベントで駅に 3 秒間停車してから発車します。

```
import time

import atovrmnxclient as vc


client = vc.Client()

ats1Ai = vc.ATS(client, 80)
ats2i = vc.ATS(client, 86)

train1 = vc.Train(client, 39)


def sequence(train):
    train.stop(550)
```

```
    time.sleep(3)
    train.start(400)


ats1Ai.forward = sequence
ats2i.forward = sequence


def main():
    thread = client.connect()

    train1.start(400)

    thread.join()


if __name__ == '__main__':
    main()
```

「 IDLE 」のインタラクティブモードで列車を動かしたときは、Client オブジェクトを生成してすぐ TCP サーバーに接続していましたが、サンプルスクリプトでは、準備が全部終わってからメイン関数の中で接続しています。

メイン関数の中で Client オブジェクトの connect() メソッドが返す「 catch イベント受信スレッド」の終了を join() で待ち続けますが、これはスクリプトをコマンドプロンプトから実行する場合の対策です。

コマンドプロンプトから実行する場合はファイルの内容を実行してすぐ Python インタープリタが終了してしまいます。catch イベント受信スレッドが終了するまで（ while True: のループなのでコマンドプロンプトを強制終了するまで）Python インタープリタを終了させたくない場合は join() で待ち続けます。

```
def main():
    thread = client.connect()

    train1.start(400)

    thread.join()
```

「 IDLE 」の Editor 画面の「 Run Module F5 」メニューをクリックしたときは、フ

ァイルの内容を実行したあとそのまま Shell 画面でインタラクティブモードを開始します。「 catch イベント受信スレッド」も動作し続けるので join() は不要です。

join() が在るとファイルの内容の実行が終了せず、インタラクティブモードが開始されません。サンプルスクリプトを「 IDLE 」で実行するとき join() の行をコメントアウトすると、スクリプト実行中でもインタラクティブモードを使うことができます。

14　ato_loopline0 スクリプト

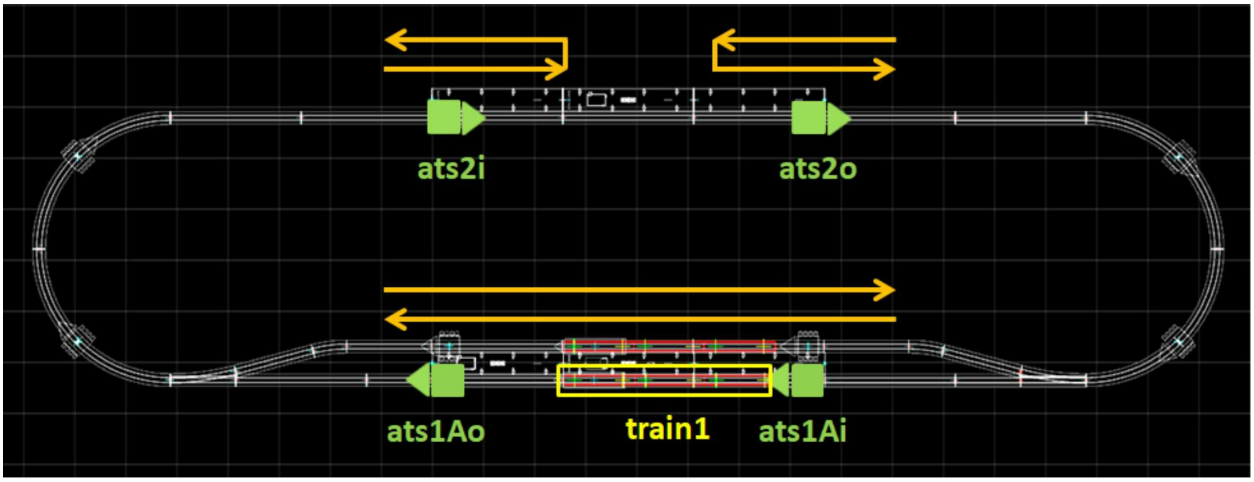
ato_loopline0 スクリプトは、1つの列車が2つの駅に停車しつつ周回します。駅の入口に配置した ATS の catch イベントで駅に 3 秒間停車してから発車します。


```
import time

import atovrmnxclient as vc


client = vc.Client()

ats1Ai = vc.ATS(client, 80)
ats2i = vc.ATS(client, 86)

train1 = vc.Train(client, 39)


def sequence(train):
    train.stop(550)
    time.sleep(3)
    train.start(400)
```

```
ats1Ai.forward = sequence
ats2i.forward = sequence


def main():
    thread = client.connect()

    train1.start(400)

    thread.join()


if __name__ == '__main__':
    main()
```

15　ato_backandforth0 スクリプト

ato_backandforth0 スクリプトは、1つの列車が2つの終着駅の間を1つの途中駅に停車しつつ往復します。それぞれの駅の入口に配置した ATS の catch イベントで駅に 3 秒間停車してから発車します。終着駅では進行方向を反転します。




```
import time

import atovrmnxclient as vc


client = vc.Client()

ats1Ai = vc.ATS(client, 80)
ats1Ao = vc.ATS(client, 81)
ats2i = vc.ATS(client, 86)
ats2o = vc.ATS(client, 87)

train1 = vc.Train(client, 39)


def sequence1(train):
```

```
    train.stop(550)
    time.sleep(3)
    train.start(400)


def sequence2(train):
    train.stop(550)
    train.Turn()
    time.sleep(3)
    train.start(400)


ats1Ai.forward = sequence1
ats1Ao.reverse = sequence1
ats2i.forward = sequence2
ats2o.reverse = sequence2


def main():
    thread = client.connect()

    train1.start(400)

    thread.join()


if __name__ == '__main__':
    main()
```

16　Platform クラス

Platform クラスは駅のプラットホームのクラスです。

codes プロパティが空な場合、進入してきたすべての列車をゆっくり停車させます。codes プロパティが空でない場合、codes プロパティに種別コードが含まれる列車と種別コードの無い列車だけをゆっくり停車させます。
コンストラクタに「 restart= 停車秒数」が渡された場合は停車秒数後ゆっくり発車させます。

16.1 コンストラクタ

def __init__(self, atses, restart=None, startdistance=400, stopdistance=550, train=None, name=None):

Args:
atses (tuple(ATS)): プラットホームの両端の ATS のタプル。
プラットホームに入る向きの ATS 、出る向きの ATS の順 (□ > 、 □ >) 。
終点で片側にしか ATS がない場合は (□ > 、None) または (None 、 □ >) のようにする。
ATS の向きと反対に走行する列車にも対応しているので、単線の往復走行にも使用できる。
restart (float): 停車した列車を指定秒数後ゆっくり発車。省略すると停車のまま。
startdistance (float): 発車時の加速距離 mm 。
stopdistance (float): 停車時の減速距離 mm 。
train (Train): 初期状態でプラットホームに入っている列車。
name (str): デバッグ用に print() で表示する名前。

16.2　メソッド

def enter(self, train):

プラットホームに列車が停車したことにする。

プラットホームに登録した ATS により自動的に呼ばれる。
引数でわたされた列車をこのプラットホームに停車中の列車とする。

Args:
  train (Train): プラットホームに停車したことにする列車。

def leave(self, train=None):

プラットホームから列車が発車したことにする。

プラットホームに登録した ATS により自動的に呼ばれる。
このプラットホームに停車中の列車は無しになる。
列車を指定した場合、指定した列車が停車中なら発車したことにする。
列車を指定しない場合、どの列車が停車中でも発車したことにする。

Args:
  train (Train): プラットホームから発車したことにする列車。

Returns:
  Train: プラットホームから発車したことにした列車。なければ None を返す。

def start(self, train=None, distance=None, voltage=None):

プラットホームから列車を発車させる。

列車を指定した場合、指定した列車が停車中なら発車させる。
列車を指定しない場合、どの列車が停車中でも発車させる。

Args:
  train (Train): プラットホームから発車させる列車。

distance (float): 発車時の加速距離。省略時は startdistance プロパティの値。
voltage (float): 走行速度の電圧。省略時は列車の voltage プロパティの値。

Returns:
Train: 発車させた列車。なければ None を返す。

16.3　プロパティ

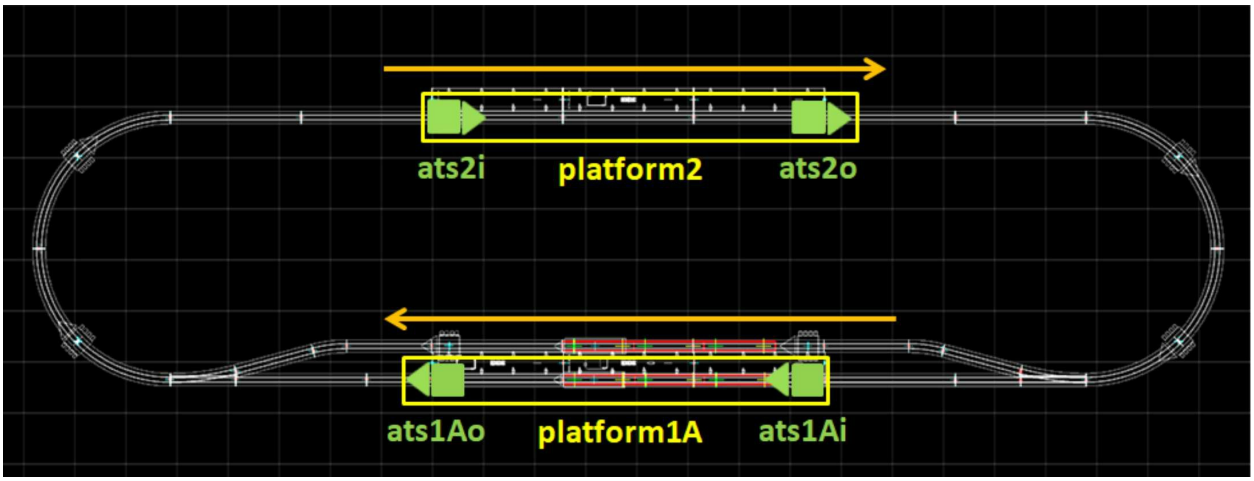
codes

プラットホームに停車させる種別コードのリストを取得する。

17　ato_loopline スクリプト

駅での停車→発車の動作を Platform オブジェクトに任せます。2つのプラットホームとも通過するタイプなので、ATS オブジェクト2つのタプルをコンストラクタに渡します。ATS の向きは揃っている必要があります。プラットホームに入る向きの ATS 、出る向きの ATS 、の順でタプルを作ります。プラットホームのどちらの向きから列車が進入しても停車→発車の動作をするので、単線の折り返し運転にも対応できます。

(ats1Ai, ats1Ao) 、(ats2i, ats2o)



3 秒間停車してから発車させたいので、「 restart=3 」をコンストラクタに渡します。

platform1A には初期状態で train1 が停車しているので、「 train=train1 」をコンストラクタに渡します。

メイン関数でプラットホームの start() メソッドを使っていますが、Train オブジェクトの start() メソッドと異なり引数無しでもゆっくり加速します。

```
import atovrmnxclient as vc


client = vc.Client()

ats1Ai = vc.ATS(client, 80)
ats1Ao = vc.ATS(client, 81)
```

```
ats2i = vc.ATS(client, 86)
ats2o = vc.ATS(client, 87)

train1 = vc.Train(client, 39)

platform1A = vc.Platform((ats1Ai, ats1Ao), restart=3, train=train1)
platform2 = vc.Platform((ats2i, ats2o), restart=3)


def main():
    thread = client.connect()

    platform1A.start()

    thread.join()


if __name__ == '__main__':
    main()
```

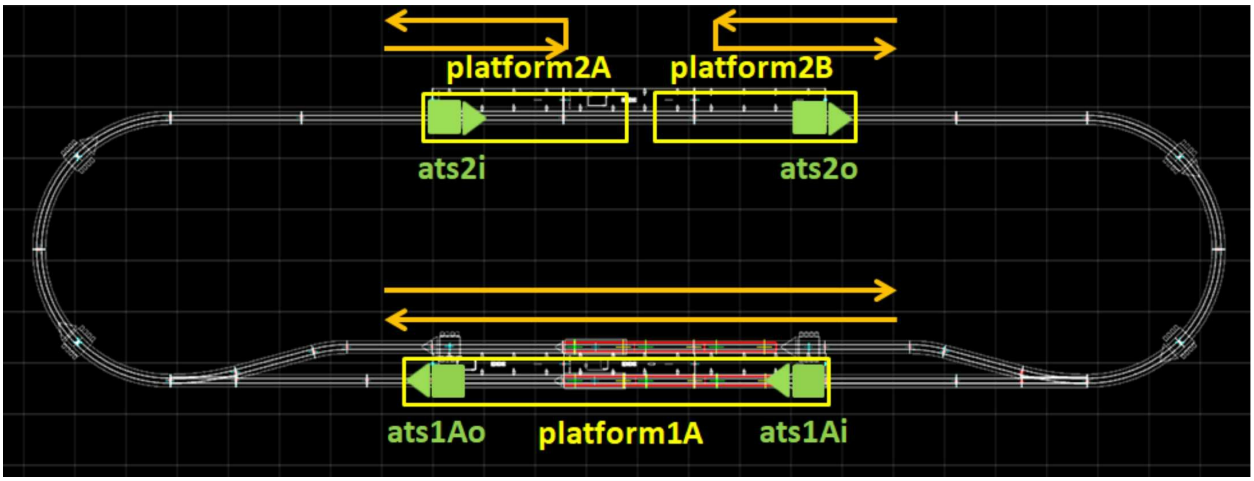
18　ato_backandforth スクリプト

終着駅での停車→反転→発車の動作も Platform オブジェクトに任せます。platform1A は通過するタイプで、ATS オブジェクト2つのタプルをコンストラクタに渡します。

(ats1Ai, ats1Ao)

platform2A 、platform2B は折り返すタイプで、タプルの ATS が無い方に「 None 」を入れます。出入口が一か所なので列車は折り返します。

(ats2i, **None**) 、(**None**, ats2o)



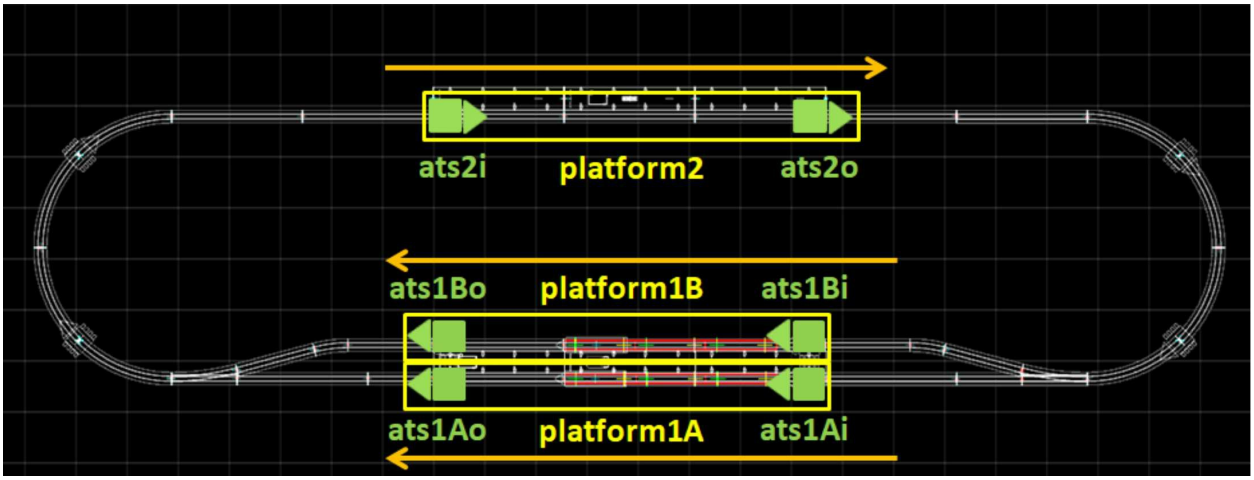

```
import atovrmnxclient as vc


client = vc.Client()

ats1Ai = vc.ATS(client, 80)
ats1Ao = vc.ATS(client, 81)
ats2i = vc.ATS(client, 86)
ats2o = vc.ATS(client, 87)

train1 = vc.Train(client, 39)

platform1A = vc.Platform((ats1Ai, ats1Ao), restart=3, train=train1)
```

```
platform2A = vc.Platform((ats2i, None), restart=3)
platform2B = vc.Platform((None, ats2o), restart=3)


def main():
    thread = client.connect()

    platform1A.start()

    thread.join()


if __name__ == '__main__':
    main()
```

19　ato_alternation スクリプト

３つのプラットホームとも通過するタイプなので、ATS オブジェクト2つのタプルをコンストラクタに渡します。

(ats1Ai, ats1Ao) 、(ats1Bi, ats1Bo) 、(ats2i, ats2o)



platform2 は停車→発車の動作を Platform オブジェクトに任せます。3 秒間停車してから発車させたいので、「 restart=3 」をコンストラクタに渡します。

platform1A と platform1B は停車の動作だけを Platform オブジェクトに任せます。停車したままにしたいので、「 restart=3 」をコンストラクタに渡しません。Platform オブジェクトの停車の処理が終わってから 3 秒後に反対のプラットホームの列車を発車させます。

```
import time

import atovrmnxclient as vc


client = vc.Client()

ats1ii = vc.ATS(client, 79)
ats1Ai = vc.ATS(client, 80)
ats1Ao = vc.ATS(client, 81)
ats1Bi = vc.ATS(client, 82)
```

```
ats1Bo = vc.ATS(client, 83)
ats2i = vc.ATS(client, 86)
ats2o = vc.ATS(client, 87)

point1 = vc.Point(client, 73)
point2 = vc.Point(client, 74)

train1 = vc.Train(client, 39)
train2 = vc.Train(client, 40)

platform1A = vc.Platform((ats1Ai, ats1Ao), train=train1)
platform1B = vc.Platform((ats1Bi, ats1Bo), train=train2)
platform2 = vc.Platform((ats2i, ats2o), restart=3)


def sequence1ii(train):
    if train == train1:
        point1.SetBranch(0)
    elif train == train2:
        point1.SetBranch(1)


def sequence1Ao(train):
    point2.SetBranch(0)


def sequence1Bo(train):
    point2.SetBranch(1)


def sequence1Ai(train):
    time.sleep(3)
    platform1B.start()


def sequence1Bi(train):
    time.sleep(3)
```

```
    platform1A.start()


ats1ii.forward = sequence1ii
ats1Ao.forward = sequence1Ao
ats1Bo.forward = sequence1Bo
ats1Ai.forward = sequence1Ai
ats1Bi.forward = sequence1Bi


def main():
    thread = client.connect()

    platform1A.start()

    thread.join()


if __name__ == '__main__':
    main()
```

20　時刻表フォーマット

これ以降は、時刻表のとおりに列車を運行するスクリプトを作成していきます。

下記のようなフォーマットの時刻表の文字列を解釈し、複数の列車を時刻表どおりに運行します。

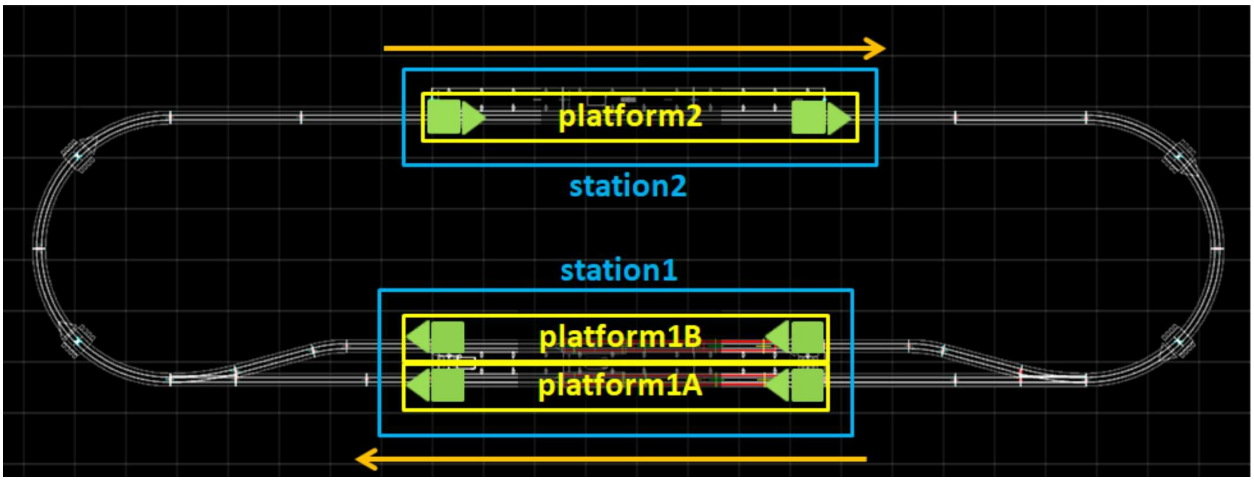
列車番号ごとに各駅の発車時刻 (分 : 秒) をカンマで区切って並べます。各行の左端の時刻が始発駅の発車時刻で、右端の時刻が終着駅に到着時刻です。時刻以外 (---->) が書かれている駅は通過します。

```
列車番号,駅１   ,駅２   ,駅１   ,駅２
A0000   ,00:00  ,00:40  ,01:10
B0040   ,00:40  ,01:20  ,01:50
A0200   ,02:00  ,----->  ,03:00
B0240   ,02:40  ,03:20  ,03:50
```

```
列車番号,駅１   ,駅２   ,駅３   ,駅４   ,駅１   ,駅２   ,駅３   ,駅４
A0000   ,00:00  ,00:40  ,01:20  ,02:00  ,02:30
B0000   ,       ,00:00  ,00:40  ,01:20  ,02:00  ,02:30
C0000   ,       ,       ,00:00  ,00:40  ,01:20  ,02:00  ,02:30
D0000   ,       ,       ,       ,00:00  ,00:40  ,01:20  ,02:00  ,02:30
```

21　時刻表スクリプトの基本構成

時刻表に書かれた駅名、列車番号に対応するため、Station オブジェクトを追加します。



これまでのサンプルスクリプトに比べて、下記黄色の部分が追加されています。

プラットホームは停車させるだけにして、新たに追加する時刻表関数が時刻表どおりに列車を発車させます。

```
パッケージをインポート

Client オブジェクト生成
ATS オブジェクト生成
Point オブジェクト生成
Train オブジェクト生成

Platform オブジェクト生成

ポイント切り替え関数定義
ATS オブジェクトにポイント切り替え関数登録

列車番号・列車紐づけ関数定義
Station オブジェクト生成
```

時刻表関数定義

メイン関数定義
メイン関数実行

「 Client オブジェクト生成」～「 ATS オブジェクトにポイント切り替え関数登録」の実際のコードは次の様になります。これまでのサンプルスクリプトとほぼ同じです。

Platform オブジェクトは自動的に列車を発車させないので、コンストラクタに「 restart= 停車時間」を渡していません。

```
import time
import datetime

import atovrmnxclient as vc


client = vc.Client()

ats1ii = vc.ATS(client, 79)
ats1Ai = vc.ATS(client, 80)
ats1Ao = vc.ATS(client, 81)
ats1Bi = vc.ATS(client, 82)
ats1Bo = vc.ATS(client, 83)
ats2i = vc.ATS(client, 86)
ats2o = vc.ATS(client, 87)

point1 = vc.Point(client, 73)
point2 = vc.Point(client, 74)

train1 = vc.Train(client, 39)
train2 = vc.Train(client, 40)

platform1A = vc.Platform((ats1Ai, ats1Ao), train=train1)
platform1B = vc.Platform((ats1Bi, ats1Bo), train=train2)
platform2 = vc.Platform((ats2i, ats2o))
```

```
def sequence1ii(train):
    if train == train1:
        point1.SetBranch(0)
    elif train == train2:
        point1.SetBranch(1)


def sequence1Ao(train):
    point2.SetBranch(0)


def sequence1Bo(train):
    point2.SetBranch(1)


ats1ii.forward = sequence1ii
ats1Ao.forward = sequence1Ao
ats1Bo.forward = sequence1Bo
```

時刻表関数はメイン関数の中で別スレッドとして実行開始させます。サンプルスクリプトでは Client オブジェクトの startthread() メソッドを使っています。

```
def main():
    thread = client.connect()

    client.startthread(timetable)

    thread.join()


if __name__ == '__main__':
    main()
```

22　Station クラス

Station クラスはプラットホームを管理する駅のクラスです。

時刻表は駅名と列車番号と発車時刻（終着駅は到着時刻）で構成されています。

Station オブジェクトは、時刻表に書かれている「駅名」、その駅に属する Platform オブジェクトのタプル、
列車番号と列車とを紐づける関数をコンストラクタに渡して生成します。
下記の例では ’A’ で始まる列車番号のときは train1 を、’B’ で始まる列車番号のときは train2 を発車させます。

```
def trainnumbertotrain(trainnumber):
    if trainnumber[0] == 'A':
        return train1
    elif trainnumber[0] == 'B':
        return train2
    else:
        return None


station1 = vc.Station(' 駅１ ', (platform1A, platform1B),
trainnumbertotrain)
station2 = vc.Station(' 駅２ ', (platform2,), trainnumbertotrain)
```

Station オブジェクトは numbers プロパティというリストを持っていて、その駅に停車させる列車の列車番号だけを追加しておきます。駅に属するプラットホームに進入してきた列車の列車番号が numbers プロパティに含まれるときだけ列車を停車させます。

時刻表の発車時刻になったなら、start(列車番号) メソッドで列車を発車させます。指定した列車番号の列車が駅に属するプラットホームに停車していた場合のみ列車を発車させることができます。

22.1　コンストラクタ

def __init__(self, name, platforms, numbertotrain):

Args:
name (str): 駅名。
platforms (tuple(Platform)): 駅に属するプラットホームのタプル。
numbertotrain (function): 列車番号に該当する列車を返す関数。
型： 関数名 (列車番号) -> 列車

22.2　メソッド

def start(self, number):

駅から列車を発車させる。

Args:
  number (str): 発車させる列車の列車番号。

Returns:
  Train: 発車させた列車。なければ None を返す。

22.3　プロパティ

name

駅名を取得・設定する。

platforms

駅に属するプラットホームのタプルを取得する。

numbers

駅に停車させる列車番号のリストを取得する。

23　ato_timetable0 スクリプト

先ずは時刻表を人が解釈し、time.sleep() で発車時刻を調整して時刻表どおりに列車を発車してみます。

Station オブジェクトのおかげで時刻表関数の中身はシンプルです。

```
def timetable():

    def starttrain(Station オブジェクト , 列車番号 ):
        Station オブジェクトの start( 列車番号 ) メソッドを実行

        Station オブジェクトの numbers プロパティに停車させる列車の列車番号を追加

    切りの良い 00 秒まで待つ

     while True:
         time.sleep() で発車時刻を調整して、上記 starttrain(Station オブジェクト , 列車番号 ) 関数を実行
```

時刻表関数の実際のコードは次の様になります。

```
def timetable():

    def starttrain(station, trainnumber):
        t = station.start(trainnumber)
        if t:
            print(f'{datetime.datetime.now()}: [{station.name}] から [{trainnumber}] 発車 ')
        else:
            print(f'{datetime.datetime.now()}: [{station.name}] に [{trainnumber}] 不在 ')

    #  列車番号　, 駅１ , 駅２ , 駅１ , 駅２
     # A0000      ,00:00  ,00:40  ,01:10
    # B0040      ,00:40  ,01:20  ,01:50
    # A0200      ,02:00  ,---->    ,03:00
```

```
    # B0240      ,02:40  ,03:20  ,03:50

    station1.numbers.extend(('A0000', 'B0040', 'A0200', 'B0240')) #  全て停車
     station2.numbers.extend(('A0000', 'B0040',          'B0240')) # A0200 通過

     # 00 秒から開始
     now = datetime.datetime.now()
    basetime = (now + datetime.timedelta(minutes=1)).replace(second=0)
    print(f' 現在時刻 : {now.strftime("%H:%M:%S")}')
    print(f' 開始時刻 : {basetime.strftime("%H:%M:%S")}')
    print()
    while datetime.datetime.now() < basetime:
        time.sleep(0.1)

    while True:
        starttrain(station1, 'A0000')
        time.sleep(40)
        starttrain(station2, 'A0000')
        starttrain(station1, 'B0040')
        time.sleep(40)
        starttrain(station2, 'B0040')
        time.sleep(40)
        starttrain(station1, 'A0200')
        time.sleep(40)
        # A0200 駅2通過
         starttrain(station1, 'B0240')
        time.sleep(40)
        starttrain(station2, 'B0240')
        time.sleep(40)
```

24　schedule パッケージ

時刻表どおりに列車を発車させるため、schedule パッケージをインポートします。

次の様に登録すると毎日同じ時刻に関数を実行します。

```
schedule.every().day.at( 時刻 ).do( 関数 , 引数 , ...)
```

下記スクリプトは次の時刻表の内容を毎日 06:00:00 ～ 06:03:20 に実行します。

```
列車番号,駅１　,駅２　,駅１　,駅２
A0000　,00:00　,00:40　,01:10
B0040　,00:40　,01:20　,01:50
A0200　,02:00　,----> ,03:00
B0240　,02:40　,03:20　,03:50
```

```
import time
import schedule

def starttrain(station, trainnumber):
    print(f'{station} から {trainnumber} 発車 ')
    station.start(trainnumber)

schedule.every().day.at('06:00:00').do(starttrain, station1, 'A0000')
schedule.every().day.at('06:00:40').do(starttrain, station2, 'A0000')
schedule.every().day.at('06:00:40').do(starttrain, station1, 'B0040')
schedule.every().day.at('06:01:20').do(starttrain, station2, 'B0040')
schedule.every().day.at('06:02:00').do(starttrain, station1, 'A0200')
schedule.every().day.at('06:02:40').do(starttrain, station1, 'B0240')
schedule.every().day.at('06:03:20').do(starttrain, station2, 'B0240')

while True:
    schedule.run_pending()
    time.sleep(1)
```

25　ato_timetable1 スクリプト

次は時刻表を人が解釈し、schedule パッケージに発車時刻を終日分登録してみます。毎日登録した時刻に実行されるので、スクリプトの実行を強制終了するまで時刻表の内容を延々と繰り返します。

```
def timetable():

    def starttrain(Station オブジェクト , 列車番号 ):
        Station オブジェクトの start( 列車番号 ) メソッドを実行

    def tohhmmss(dt, mmss, log):
        時刻表の相対時刻 (MM:SS) を実際の時刻 (HH:MM:SS) に変換して返す

        Station オブジェクトの numbers プロパティに停車させる列車の列車番号を追加

    切りの良い 00 秒を開始時刻とする
    翌日の発車時刻で動作しないように、登録を開始時刻の 5 秒前まで待つ

    時刻表が 4 分間分なので、4 分間隔で終日分繰り返し登録
        上記 tohhmmss() 関数で変換した発車時刻ごとに、
            上記 starttrain() 関数、Station オブジェクト , 列車番号を schedule パッケージに登録

    登録した starttrain() 関数が実行されるように schedule パッケージの run_pending() を実行し続ける
```

schedule パッケージをインポートします。

```
import schedule
```

時刻表関数の実際のコードは次の様になります。

```
def timetable():

    def starttrain(station, trainnumber):
```

```
        t = station.start(trainnumber)
        if t:
            print(f'{datetime.datetime.now()}: [{station.name}] から
[{trainnumber}] 発車 ')
        else:
            print(f'{datetime.datetime.now()}: [{station.name}]
に [{trainnumber}] 不在 ')

    def tohhmmss(dt, mmss, log):
        zero = datetime.datetime.strptime('00:00', '%M:%S')
        delta = datetime.datetime.strptime(mmss, '%M:%S') - zero
        hhmmss = (dt + delta).strftime('%H:%M:%S')
        if log:
            print(f'{mmss} -> {hhmmss}')
        return hhmmss

    #  列車番号   , 駅１ , 駅２ , 駅１ , 駅２
     # A0000      ,00:00  ,00:40  ,01:10
    # B0040      ,00:40  ,01:20  ,01:50
    # A0200      ,02:00  ,---->    ,03:00
    # B0240      ,02:40  ,03:20  ,03:50

    station1.numbers.extend(('A0000', 'B0040', 'A0200', 'B0240')) #  全て停
車
     station2.numbers.extend(('A0000', 'B0040',          'B0240')) # A0200 通
過

     schedule.clear()

    # 00 秒から開始
     now = datetime.datetime.now()
    basetime = (now + datetime.timedelta(minutes=1)).replace(second=0)
    if now.second >= 60 - 5: #  登録に 5 秒確保
         basetime = (now +
datetime.timedelta(minutes=2)).replace(second=0)
    print(f' 現在時刻 : {now.strftime("%H:%M:%S")}')
    print(f' 開始時刻 : {basetime.strftime("%H:%M:%S")}')
```

```
    print()

    #  翌日の発車時刻で動作しないように、登録を開始時刻の 5 秒前まで待つ
     waittime = basetime - datetime.timedelta(seconds=5)
    while datetime.datetime.now() < waittime:
        time.sleep(1)

    print(f' 登録開始 : {datetime.datetime.now().strftime("%H:%M:%S")}')

    # 4 分間隔で終日分繰り返し登録
     minutes = 4
    for m in range(0, 24 * 60 - (minutes - 1), minutes):
        log = m == 0 or m == 24 * 60 - minutes
        if m == minutes:
            print('...')

        t = basetime + datetime.timedelta(minutes=m)

        schedule.every().day.at(tohhmmss(t, '00:00', log)).do(starttrain, station1, 'A0000')
        schedule.every().day.at(tohhmmss(t, '00:40', log)).do(starttrain, station2, 'A0000')

        schedule.every().day.at(tohhmmss(t, '00:40', log)).do(starttrain, station1, 'B0040')
        schedule.every().day.at(tohhmmss(t, '01:20', log)).do(starttrain, station2, 'B0040')

        schedule.every().day.at(tohhmmss(t, '02:00', log)).do(starttrain, station1, 'A0200')
        # A0200 駅2通過

         schedule.every().day.at(tohhmmss(t, '02:40', log)).do(starttrain, station1, 'B0240')
        schedule.every().day.at(tohhmmss(t, '03:20', log)).do(starttrain,
```

```
station2, 'B0240')

    print(f' 登録完了：
{datetime.datetime.now().strftime("%H:%M:%S")}')
    print()

    while True:
        schedule.run_pending()
        time.sleep(1)
```

26　ato_timetable スクリプト

atovrmnxclient モジュールは、時刻表の文字列を解釈して schedule パッケージに登録する関数が実装されています。

def cleartimetable(secondsago=5):

schedule パッケージをクリアして自動運転の開始時刻を返す。

Args:
　secondsago (float): 自動運転の開始時刻の何秒前に登録するか。

Returns:
　datetime.datetime: 自動運転の開始時刻。

def readtimetable(basetime, minutes, timetable, stations, starttrain):

時刻表の文字列を解釈して各駅の発車時刻を schedule パッケージに登録する。

下記フォーマットの文字列を解釈して、各駅の発車時刻を schedule パッケージに登録する。
また、駅に停車させる列車番号を各駅の numbers プロパティに追加する。

列車番号 , 駅１　, 駅２ , 駅１ , 駅２
A0000,00:00 ,00:40 ,01:00
B0000,　,00:00 ,01:20 ,01:50
A0200,02:00 ,---->　,03:10
B0200,　　,02:00 ,03:20 ,03:50

発車時刻を minutes 周期で繰り返し終日分登録する。
例 :
　00:00 => 09:20:00, 09:24:00, ..., 23:56:00, 00:00:00, ..., 09:12:00, 09:16:00
　00:40 => 09:20:40, 09:24:40, ..., 23:56:40, 00:00:40, ..., 09:12:40,

```
09:16:40

    Args:
        basetime (datetime.datetime):  自動運転の開始時刻。
         minutes (int):  時刻表の文字列に記述されている分数。
         timetable (str):  時刻表の文字列。
         stations (Station):  駅のタプル。
         starttrain (function):  駅から列車番号の列車を発車させる関数。
           型 :  関数名 ( 駅 ,  列車番号 )
```

時刻表関数の実際のコードはシンプルになります。時刻表を文字列にして readtimetable() に渡しています。

```
 def timetable():

    def starttrain(station, trainnumber):
        t = station.start(trainnumber)
        if t:
            print(f'{datetime.datetime.now()}: [{station.name}] から
[{trainnumber}] 発車 ')
        else:
            print(f'{datetime.datetime.now()}: [{station.name}]
に [{trainnumber}] 不在 ')

    basetime = vc.cleartimetable()

    timetabletext = """
 列車番号 , 駅1, 駅2 , 駅1 , 駅2
 A0000  ,00:00  ,00:40  ,01:10
 B0040  ,00:40  ,01:20  ,01:50
 A0200  ,02:00  ,---->    ,03:00
 B0240  ,02:40  ,03:20  ,03:50
 """
    stations = (station1, station2)
    vc.readtimetable(basetime, 4, timetabletext, stations, starttrain)   # 4 分
間隔で終日分繰り返し登録

     while True:
```

```
        schedule.run_pending()
        time.sleep(1)
```

この ato_timetable スクリプトが「時刻表どおりに複数の列車を運行させる Python スクリプト」の最終版です。

レイアウトに合わせて、オブジェクト、ポイント切り替え関数等を追加し、列車番号・列車紐づけ関数を変更してください。

```
パッケージをインポート

Client オブジェクト生成
ATS オブジェクト生成
Point オブジェクト生成
Train オブジェクト生成

Platform オブジェクト生成

ポイント切り替え関数定義
ATS オブジェクトにポイント切り替え関数登録

列車番号・列車紐づけ関数定義
Station オブジェクト生成

時刻表関数定義

メイン関数定義
メイン関数実行
```

あとは時刻表文字列を色々書き換えて自動運転を楽しんでください。