CTC
Challenging Tomorrow's Changes

明日からはじめる ネットワーク運用自動化 始動編

2018/1/9 10:00-12:00JST

ITOCHU Techno-Solutions America, Inc.
NetOpsCoding Committee
Taiji Tsuchiya / 土屋 太二

概要

- 本発表の概要ページ
- 2018年1月25日(木) に以下の発表を行います。

  JANOG41 明日からはじめるネットワーク運用自動化

  本発表では、上記内容のうち、プログラミングの基本部分にフォーカスして紹介します。
- 関連情報:
  - 発表者インタビュー：明日からはじめるネットワーク運用自動化
  - JANOG41 ネットワーク運用自動化BoF

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

自己紹介

- 土屋 太二/Taiji Tsuchiya
- Career
  - Solution Engineer @ CTC America (2017.8-Present)
    - 日系企業の技術支援
    - グローバル企業のネットワーク/インフラ技術の調査
    - アプリケーションソフトウェアのプロトタイプ開発
  - Network Engineer @ BIGLOBE (2011.4-2017.8)
    - DC/バックボーン/Peeringの運用/設計/開発
    - ネットワーク運用自動化システム、SDNシステムの開発
- Community Activities
  - NetOpsCoding 運営委員
  - 過去JANOG プログラム委員, 実行委員長など

Copyright © 2018 ITOCHU Techno-Solutions
America, Inc. CONFIDENTIAL ONLY

インフラ運用の自動化への要望
(ネットワークに限らず)

- 機器作業における人為ミスを無くしたい。
- 運用に関連する作業の効率性を高めたい。
  - 例: 機器一台あたりの作業/運用にかかる時間を短縮したい。
  - 例: 技術者一人あたりの運用対象機器を増やしたい。
- 障害発生時に、迅速に原因を発見および復旧したい。
- 障害発生前に、予兆を発見し、危険因子を取り除きたい。
- サービス開発チームからの要求に応じて、
  迅速にインフラの準備・更新・廃止を実施したい。
- インフラの正常な状態を維持したい、
  もしくは異常発生時に迅速に正常状態を再現したい。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

ネットワーク運用現場から聞こえる声

自動化が必要なのはわかる。
ただ、どのように取り組むべきかわからない…

自動化開発のための知識・スキルが無い…

ネットワークと開発、両方できる人材がいない…

自動化に対する組織の理解・サポートが足りない…

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

運用自動化の第一歩目:
まずは運用現場からはじめる

運用課題(小)
- 手早く改善効果を出せる作業
- 定型化しやすい作業
- 作業影響が無い作業
- 開発規模が小さいもの

Step.1
- 運用メンバ数名ではじめる
- 影響の少ない作業から自動化
- ソフトウェアを小さく作る
- 社内での実績と協力者を作る

運用課題(中)
- 改善効果が中程度の作業
- 作業影響があるが小~中程度
  もしくは手動復旧できる作業
- 開発規模が中程度のもの

Step.2
- 開発専任メンバーを立てる
    - 運用メンバから抜擢する
    - 他チームから連れてくる
    - 社外から連れてくる
- ソフトウェアを疎結合に作る
- ドキュメントを残す
- 組織の理解と信頼を得る

運用課題(大)
- 改善効果が大きい作業
- 作業影響が大きい作業
- 開発規模が大きいもの
- 継続的に開発が必要なもの

Step.3
- 開発チームを立ち上げる
- 運用性・可用性・スケール
  を考慮したシステムを構築する
- 運用と開発が密に連携しながら
  課題解決に継続的に取り組む
- 組織として、システムと開発体制
  を維持・安定させる

本発表のねらい

- ネットワーク運用現場のエンジニア自身が「明日から」自動化開発を始めるための手段を共有。
  - 具体的なプログラミングテクニック
  - 開発に便利なOSSライブラリ・ツール
  - 自動化サンプルコード
- 本発表での知見を、皆様の職場に持ち帰っていただき、運用改善のための議論をしていただきたいです。
  - 今の運用業務のなかで「自動化」「効率化」「省力化」ができるところはないか？
  - 「どの業務を」「どの手段で」「誰が 」「どこから」着手できそうか？
  - 組織・チームとしてサポートできることはないか？

ネットワーク運用自動化の一例

- ネットワーク装置の設定の自動化
  - インタフェース, ACL, BGP, Route Policy, MPLS LSP
- 作業手順書における作成工程の自動化・省力化
- リソース情報管理のDB化・脱Excel化
- ネットワークトポロジー管理の見える化・省力化
- 監視ツール運用の自動化・省力化
- 大量のネットワーク装置へのコマンド同時実行・情報取得
- ネットワーク障害の自動検知/自動復旧
- ネットワーク装置の機能強化
  - CLI、API、Routing機能などの
自社運用に最適な形にカスタマイズ

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

ネットワーク自動化 失敗パターン by Kirk Byers

1. ハイリスクな問題、難しい問題から手をつける 。
2. All or Nothingなマインドセットで考える(全工程自動化 vs 全手動)
3. すべてを自分自身で再発明しようとする。
4. 理解なしで成功パターン&コードの表面的なコピーで済ませようとする
5. 良いデバッグ方法を学ばない。
6. Over-Engineeringな方法で解決しようとする。
(不必要なことまで技術で解決しようとする)
7. 小さなスケールから学ばない。
8. 忙しすぎて自動化する時間が無い。
9. コードの再利用の方法を学ばない。 (長期的な観点)
10. 有用な開発者ツールを使わない。 (長期的な観点)
Git, 文法チェッカー, Unit Test, CIツールなど。

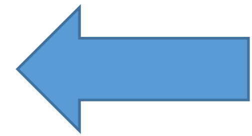
NANOG71:Network Automation: past, present, and future (Slide, Movie)

目次

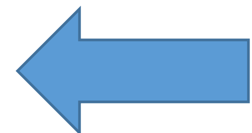
本発表では、プログラミング言語としてPython(3系)を例にします。

- Python 基礎編
    - print
    - 変数 (数値型、文字列型、リスト型、辞書型)
    - if
    - for
    - ファイル入出力
    - 関数
    - クラス
- Python 応用編
    - ライブラリ
    - JSON
    - テンプレートエンジン
    - 正規表現
    - ユニットテスト
- ネットワーク自動化開発に便利なOSSライブラリ/ツールの紹介
- やってみようネットワーク自動化
    - プログラムからルータにログイン、showコマンドを実行
    - プログラムからルータにコンフィグ投入
- 自動化サンプル: BGP Peering作業を自動化する

事前オンライン発表
の範囲
(今回の対象)



JANOG41
の範囲
(今回は対象外)



(参考)本発表の理解を深めるツール

- janog41-auto-taiji
- 本発表で紹介するサンプルコードを
  Webブラウザ上で実行することができます。
- ソースコードの変更->実行ができるので、
  試行錯誤しながら理解していくことができます。
- 本ツールは長久 勝 さん (国立情報学研究所クラウド基盤研究開発センター)
  によって開発していただきました。

- 本発表で紹介したサンプルコードはすべて公開しています。
  https://github.com/taijiji/NetworkAutomationTutorial

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

なぜPython？

- コードの記述量が少なく、書きやすく読みやすいため
  プログラミング初心者が習得しやすい。
- Webアプリ、サーバサイド、機械学習やデータ解析
  などの分野で、世界中で人気と実績がある。
  (多用途で使えるので、覚えておいて損はしない。)
- ネットワーク自動化系をはじめ、
  多くのライブラリやフレームワークが用意されている。
  (自身でゼロからコードを書く必要が無い。)

(※) 必ずしもPythonを使う必要はなく、
チームメンバー共通で既に使い慣れた
プログラミング言語があればそちらでもOKです。
ただし開発対象の関連ライブラリの有無については
開発スピードに直結するので要事前調査。

Copyright © 2018 ITOCHU Techno-Solutions
America, Inc. CONFIDENTIAL ONLY

Python 2系 vs 3系

- Python2系(最新 2.7.14)と3系(最新 3.6.3)は互換性なし。書式も差異あり。
- Python2系は「2.7」が最後のメジャーリリースであると2010年にアナウンス。
- 一部ライブラリで、Python2系しか対応してないケースあり。(2018年現在では、大半のライブラリは両対応済)
- 一部ライブラリで、Python2系のサポートを2018年頃から終了するアナウンスあり。

Python2系->3系の書き換えは骨が折れるので
今から開発をはじめるのであればPython3系がおすすめです。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

Python バージョンの調べ方

```
$ python --version

Python 2.7.14
```

デフォルトで動作している
Python バージョン
(OSによって異なる)

```
$ python2 --version

Python 2.7.14
```

```
$ python3 --version

Python 3.6.3
```

OS環境によっては、
”python2”や”python3”, ”python3.6”
のコマンドが
デフォルト設定されている場合あり。

本発表では、“python3” コマンドを利用します。

Copyright © 2018 ITOCHU Techno-Solutions
America, Inc. CONFIDENTIAL ONLY

Python3.6 インストール

- OSごとのインストール例
  - Windows 10 の場合
    - https://www.python.org/downloads/
  - Mac OS X (macOS) の場合
    - https://qiita.com/ms-rock/items/6e4498a5963f3d9c4a67
  - CentOS 7 の場合
    - https://qiita.com/estaro/items/5a9e4eb8f0902b9977e3
- バージョンがうまく切り替わらない場合は、環境変数PATHの設定変更が必要。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

Pythonプログラムの作り方

- プログラム作成
  - 拡張子「.py」をつけたファイルを作成。
  - 入れ子構造は「:」とインデント(スペース4文字 or タブ文字) で表現。
  - 他プログラム言語での、入れ子を示す「{}」や文末の「;」などは使わない。

```
$ vi sample.py

num = 1
if num == 1:
    print(“Hello JANOG!!!”)
```

スペース4文字

- プログラム実行
  - 前述のpythonコマンドで、pythonファイルを指定して実行する。

```
$ python3 sample.py

Hello JANOG!!!
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

Python 基礎編

- print： 標準出力
- 変数
  - 数値型
  - 文字列型
  - リスト型
  - 辞書型
- if： 条件分岐処理
- for :繰り返し処理
- ファイル入出力
- 関数
- クラス

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

print: 標準出力

プログラム本文

ファイル名: sample_print.py

```
print("Hello JANOG!")
```

プログラム実行結果

プログラム実行結果

```
$ python3 sample_print.py

Hello JANOG!
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

変数 (数値型)

[書式] 変数名 = 数字

プログラム本文

ファイル名: sample_number.py

```
a = 1   # a: int型として生成
b = 1.1 # b: float型として生成
print(a)
print(b)

c = a + b # c: float型として生成
print(c)
```

プログラム実行結果

```
$ python3 sample_number.py

1
1.1
2.1
```

型の指定(C言語でいう「int a;」「float b;」など)は必須ではなく、自動で型が判定されます。(動的型付け)

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

変数(文字列型)

[書式] 変数名 = ”文字列”

プログラム本文

sample_string.py

```
c = “text” # 「‘」「“」のどちらでも可。特別な違いはなし。
d = "テキスト"
#日本語文字列について
#Python3ではデフォルトでUnicode文字列として扱われる。
#Python2ではデフォルトはASCIIであるため「u"テキスト"」と明示する必要あり。

print(c)
print(d)
```

プログラム実行結果

```
$ python3 sample_string.py

text
テキスト
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

変数(リスト型)

[書式]
変数名 = [変数1, 変数2, …]

sample_list.py

```
router_list = ["router_A", "router_B", "router_C"]

# リスト型変数を出力
print(router_list)

# 0から数えて、1番目の値を出力
print(router_list[1])

# 配列の長さを取得(len)
print(len(router_list))
```

プログラム実行結果

```
$ python3 sample_list.py

['router_A', 'router_B', 'router_C']
router_B
3
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

変数(辞書型)

[書式]
変数名 = { キー1:バリュー1, キー2:バリュー2, … }

sample_dictionary.py

```
# 「key」と「value」のペアを変数として定義
router_info = {
    "hostname" : "router_A",
    "os"       : "junos",
    "version"  : "15.1",
}

# ホスト名を表示
print(router_info["hostname"])
# バージョンを表示
print(router_info["version"])
```

プログラム実行結果

```
$ python3 sample_dictionary.py

router_A
15.1
```

America, Inc. CONFIDENTIAL ONLY

if：条件分岐

変数 num の値に応じて、処理を変更する例

sample_if.py

```
num = 2

if num == 1:
    print("num is one")
elif num == 2:
    print("num is two")
else:
    print("num is the others")
```

プログラム実行結果

```
$ python3 sample_if.py

num is two
```

※Pythonには、他プログラミング言語でいうcase文は存在しない。
　すべてif文で表現。

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

for: 繰り返し処理

ある処理を、3回分繰り返す例

sample_for1.py

```
# 回数を指定して繰り返し処理を実施
# range(3) = 0,1,2が格納
for num in range(3):
    print(num)
```

プログラム実行結果

```
$ python3 sample_for1.py

0
1
2
```

ある処理を、リスト変数の数だけ繰り返す例

sample_for2.py

```
# リスト型変数を利用して繰り返し処理を実施
router_list = [”router_A”, ”router_B”]

# router_listの値を、順次 router_name に代入
for router_name in router_list:
    print(router_name)
```

プログラム実行結果

```
$ python3 sample_for2.py

router_A
router_B
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

ファイル入出力

ファイル読み込み

[ファイル名: sample_read.txt]

```
Hello JANOG! from sample_read.txt
How are you feeling?
```

ファイル読み込み

sample_file.py

```
# 読み込みモード:"r"
file1 = open("sample_read.txt", "r")
text = file1.read() # ファイル全文を文字列として読み込み
print(text)
file1.close()
```

プログラム実行結果

```
$ python3 sample_file.py

Hello JANOG! from sample_read.txt
How are you feeling?
```

ファイル書き込み

```
# 上書き書き込みモード:"w", 追記書き込みモード:"a"
file2 = open("sample_write.txt", "w")
file2.write("Good!")
file2.close()
```

ファイル書き込み

[ファイル名: sample_write.txt]

```
Good!
```

関数とは

- 関数 = 機能をひとまとまりにしたもの。
  - 例: IPv4アドレスの計算をする関数
  - 例: 各ルータOSに応じたコマンドを取得する関数
- プログラムの中で汎用的な処理を関数にして使い回すことで全体のプログラム記述量を減らすことができる。

[書式]

```
def 関数名(引数…)
    関数の処理
    return 戻り値 # 戻り値が無い場合は記述不要


# 関数の呼び出し
関数名()
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

関数 記述例

プログラム本文

sample_function.py

```
# 関数の定義
# ルータ情報を受けとり、
# OSに応じたshow bgp summaryコマンドを返す関数
def get_show_bgp_summary(router_info):
    if router_info["os"] == "junos":
        command = "show bgp summary"
    elif router_info['os'] == 'ios':
        command = "show ip bgp summary"
    else:
        command = “N/A”
    return command

# 辞書型変数
router_info = {
    "hostname"  : "router_A",
    "os"        : "junos"
}
# router_infoを引数として、関数を呼び出し
command = get_show_bgp_summary(router_info)
print(command)
```

プログラム実行結果

```
$ python3 sample_function.py

show bgp summary
```

クラスとは

- クラス = 変数や関数の定義をひとまとまりにした雛形。
- インスタンス = 雛形を元に作成するオブジェクト。
  クラスで定義した機能をもつ。
- 例: ルータクラス
  - 変数: ホスト名, OS名、version名, IPアドレス…
  - 関数: ルータ実機への接続機能、showコマンド実行機能…

[書式]

```
class クラス名:
    クラス変数
    def 関数名(引数…)
        関数の処理
        return 戻り値

# インスタンスの定義
変数名 = クラス名()
# クラス変数の呼び出し
変数名.クラス変数
# 関数の呼び出し
変数名.関数名()
```

クラス 記述例

プログラム本文

```
class Router:
    # 初期化関数: インスタンス生成時に必ず呼ばれる関数。
    def __init__(self, hostname, os, version):
        # インスタンス変数: 各インスタンスで利用する変数。
        self.hostname = hostname
        self.os = os
        self.version = version

    # show bgp summaryコマンドを返す関数。
    def get_show_bgp_summary(self):
        if self.os == "junos":
            command = "show bgp summary"
        elif self.os == 'ios':
            command = "show ip bgp summary"
        else:
            command = 'N/A’
        return command
```

```
# Routerクラスのインスタンスを生成
router_A = Router(hostname="router_A", os="junos", version="15.1")
router_B = Router(hostname="router_B", os="ios", version="15.7(3)M")

# それぞれのインスタンスでRouterクラスで定義された関数を呼び出し
print(router_A.hostname)
command = router_A.get_show_bgp_summary()
print(command)

print(router_B.hostname)
command = router_B.get_show_bgp_summary()
print(command)
```

プログラム実行結果

```
$ python3.6 sample_class.py

router_A
show bgp summary
router_B
show ip bgp summary
```

Python 応用編

- ライブラリの利用
- JSONファイルの利用
- テンプレートエンジンの利用
- 正規表現の利用
- ユニットテストの利用

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

Pythonライブラリ

第三者によって開発されたプログラムを
再利用するための仕組み。

- 標準ライブラリ
  - デフォルトでインストールされているライブラリ
  - 例: datetime, ipaddress, json

- 外部ライブラリ
  - コミュニティによって作成された外部ライブラリ
  - パッケージ管理システム「pip 」を利用した
    ライブラリのインストールが必要
  - 例: jinja2, django, napalm

Copyright © 2018 ITOCHU Techno-Solutions
America, Inc. CONFIDENTIAL ONLY

pip の利用例

外部ライブラリのインストール
(jinja2ライブラリをインストールする例)

```
$ pip3 install jinja2
```

インストール済ライブラリの一覧表示

```
$ pip3 list

Jinja2 (2.9.6)
. . .
```

(※) 応用編として、venvやvirtualenv等を利用することでアプリ開発環境ごとにpip環境を分ける方法もあります。
(アプリと必要ライブラリを明確にする意図など)
https://docs.python.jp/3/library/venv.html

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

pip のバージョンの調べ方

```
$ pip --version

pip 9.0.1 from
/Library/Frameworks/Python.framework/Versions
/2.7/lib/python2.7/site-packages (python 2.7)
```

デフォルトで動作している
pip バージョン。
対応Pythonバージョンに注目。

```
$ pip2 --version

pip 9.0.1 from
/Library/Frameworks/Python.framework/Versions
/2.7/lib/python2.7/site-packages (python 2.7)
```

```
$ pip3 --version

pip 9.0.1 from
/Library/Frameworks/Python.framework/Versions
/3.6/lib/python3.6/site-packages (python 3.6)
```

OS環境によっては、
"pip2"や"pip 3"のコマンドが
標準設定されている場合あり。

python3 –m pip コマンドでも
代替可。

(※) 参考: pip インストール方法
Pythonで一番最初に入れるべきパッケージ **setuptools** と **pip**
http://www.lifewithpython.com/2012/11/Python-package-setuptools-pip.html

ライブラリの利用例

datetimeライブラリにより現在時刻を表示

sample_library.py

```
# datetime ライブラリのインポート
import datetime

# datetimeライブラリの、datetimeクラスのnow関数を実行。
now = datetime.datetime.now()
print(now)
```

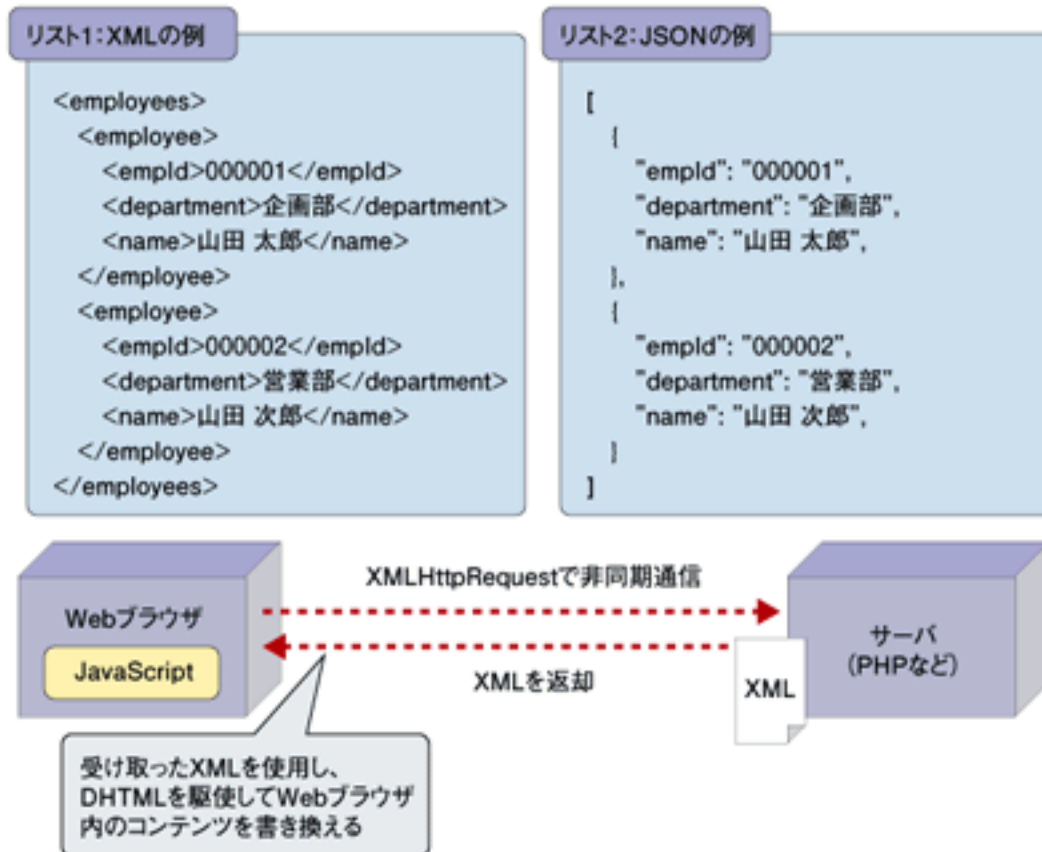
プログラム実行環境

```
$ python3 sample_library.py

2017-11-15 14:25:56.360603
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

JSONとは

データフォーマット(記法)の一つ。
プログラム間のデータやりとりやAPIで多く利用される。
従来のXMLと比較して、人間にもプログラムにも読みやすい利点がある。



参考: JSONってなにもの？
https://thinkit.co.jp/article/70/1

JSONの利用

JSON形式ファイル
(この時点ではただのテキストデータ)

sample_json.json

```
[
  {
    "router_name": "Router_A",
    "ip": "192.168.0.1",
    "os": "junos"
  },
  {
    "router_name": "Router_B",
    "ip": "192.168.0.2",
    "os": "ios"
  },
  {
    "router_name": "Router_C",
    "ip": "192.168.0.3",
    "os": "iosxr"
  }
]
```

Jsonファイルを読み込み、Python変数に変換
(この例では、辞書型変数を格納したリスト型変数 に変換)

```
import json

file = open("sample_json.json", "r")

# fileオブジェクトからjsonを読み込み、変数に格納
routers_list = json.load(file)

print(routers_list)
print(routers_list[0]["ip"])
```

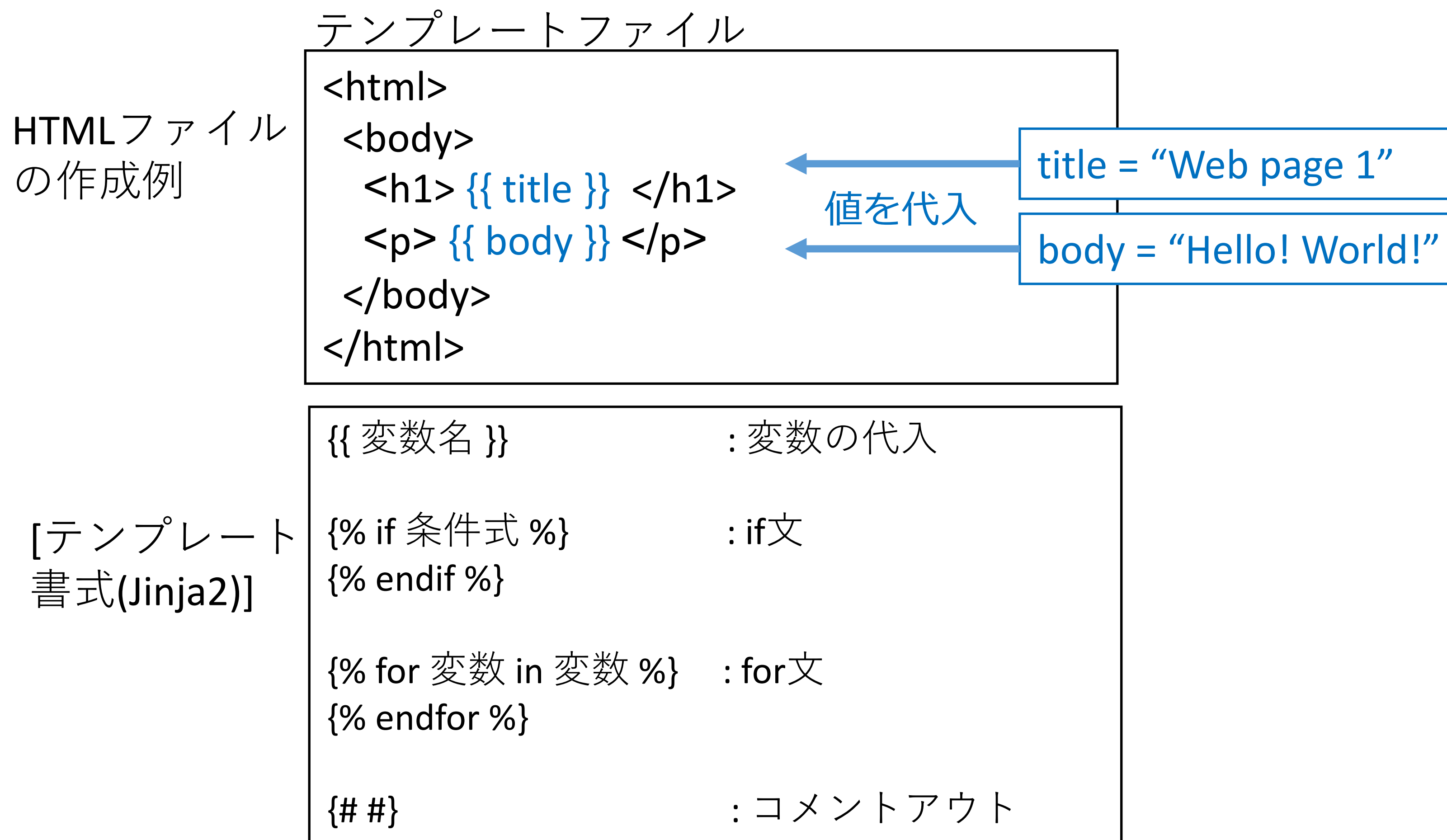
プログラム実行結果

```
$ python3 sample_json.py

[{'router_name': 'Router_A', 'ip': '192.168.0.1', 'os': 'junos'},
{'router_name': 'Router_B', 'ip': '192.168.0.2', 'os': 'ios'},
{'router_name': 'Router_C', 'ip': '192.168.0.3', 'os': 'iosxr'}]

192.168.0.1
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

テンプレートエンジンとは

テンプレートファイル(テキストの雛形)とデータモデルや変数を組み合わせて新たなテキストファイルを自動作成するモジュール。
PythonではテンプレートエンジンとしてJinja2が利用されることが多い。



テンプレートエンジンの利用

ルータコンフィグの作成

テンプレートファイル(Jinja2形式)

sample_template.jinja2

```
interface {{ if_name }}
 description {{ if_description }}
 ip address {{ ip4 }} {{ ip4_subnet }}
 duplex auto
 speed auto
 no shutdown
!
```

プログラム本文

sample_template.py

```
import jinja2

# ファイルの読み込み
file = open("sample_template.jinja2", "r")
template_txt = file.read()

# テンプレートオブジェクトの作成
template = \
jinja2.Environment().from_string(template_txt)

# テンプレートに値を代入
config_txt = template.render(
    if_description="To_RouterA",
    if_name="fastethernet 1/1",
    ip4=“192.168.0.1”,
    ip4_subnet="255.255.255.0")

print(config_txt)
```

プログラム実行結果

```
$ python3 sample_template.py

interface fastethernet 1/1
 description To_RouterA
 ip address 192.168.0.1 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

テンプレートエンジンの利用(複数作成)

テンプレートファイル(Jinja2形式)

```
interface {{ if_name }}
 description {{ if_description }}
 ip address {{ ip4 }} {{ ip4_subnet }}
 duplex auto
 speed auto
 no shutdown
!
```

プログラム本文の一部抜粋

```
interfaces = [
    {
        "if_description" : "To_RouterA",
        "if_name"        : "fastethernet 1/1",
        "ip4"            : "192.168.0.1",
        "ip4_subnet"     : "255.255.255.0"
    },
    {
        "if_description" : "To_RouterB",
        "if_name"        : "fastethernet 1/2",
        "ip4"            : "192.168.1.1",
        "ip4_subnet"     : "255.255.255.0"
    },
    {
        "if_description" : "To_RouterC",
        "if_name"        : "fastethernet 1/3",
        "ip4"            : "192.168.2.1",
        "ip4_subnet"     : "255.255.255.0"
    }
]

for interface in interfaces:
    config_txt = template.render(interface)
    print(config_txt)
```

プログラム実行結果

```
interface fastethernet 1/1
 description To_RouterA
 ip address 192.168.0.1 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
interface fastethernet 1/2
 description To_RouterB
 ip address 192.168.1.1 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
interface fastethernet 1/3
 description To_RouterC
 ip address 192.168.2.1 255.255.255.0
 duplex auto
 speed auto
 no shutdown
!
```

正規表現

パターン文字列を利用して、
文字列の集合の中から、対象の文字列を抽出する方法。

```
text    = “Today is Sunday”

pettern = ”Today is (.+)”
```

「()」：
グループ化

「+」：
直前の正規表現に対して
１回以上の繰り返し

「.」：
改行以外の
任意の文字列1文字

```
match = re.search(pettern, text )
# マッチした行全体を取得
match.group(0) # =>「Today is Sunday」
# マッチしたグループを取得
match.group(1) # =>「Sunday」
```

正規表現の利用

```
import re

# 複数行の文字列
# show versionコマンド実行結果を想定
show_version_txt = """
Hostname: vsrx
Model: firefly-perimeter
JUNOS Software Release 12.1X47-D15.4
"""

# 正規表現
# 「()」: グループ化
# 「.」 : 改行以外の任意の文字列1文字
# 「+」 : 直前の正規表現に対して、1回以上の繰り返し
regex = "JUNOS Software Release (.+)"

# show_version_txtの文字列に対して、正規表現による文字列の抽出を実行
match = re.search(regex, show_version_txt )

# マッチした文字文字列全体を表示
print(match.group(0)) # => 「JUNOS Software Release 12.1X47-D15.4」が表示

# マッチした文字列のうちグループ化された部分を抽出
print(match.group(1)) # => 「12.1X47-D15.4」が表示
```

ユニットテストとは

- ソフトウェアにおける単体テストを実施するためのモジュール。
- １つの関数に対して「入力値」と「期待する出力値」を定義することで、関数の実装の誤りを検出する。
- テストケースを書くメリット:
  - バグの早期発見が可能。
  - 関数単位で動作確認できるため、プログラム全体を動作させることなく、実装->テスト->修正->テスト->…を素早く実施することが可能。
  - 関数の振る舞いを明記することで、コードの変更・改良が容易になる。第三者もコードの意図を理解しやすくなる。
- テストケースを書くデメリット:
  - テストケースを書くことは、常に工数とのトレードオフの関係にある。
  - 異常系テストケースをどこまで実装すべきかは、プロジェクトの段階や開発対象機能に応じて検討する。

ユニットテスト 記述方法

関数の例

```
def hello(num):
    if num == 1:
        message = “Hello”
    elif num == 2:
        message = “Good bye”
    else:
        message = “No message”

    return message
```

テストケースの一例

```
# hello(1) のときは、
# 戻り値が”Hello”であるべき。
assertEqual(hello(1), “Hello”)
# hello(2) のときは、
# 戻り値が“Good bye”であるべき。
assertEqual(hello(2), “Good bye”)
# hello(-1)のときは、
# 戻り値が”No message”であるべき
assertEqual(hello(-1), “No message”)
# hello(“abc”) のときは、
# 戻り値が”No message”であるべき
assertEqual(hello(“abc”), “No message”)
```

ユニットテスト 利用例

テスト対象のプログラム

sample_unittest.py

```python
def add(x, y):
    sum = x + y
    return sum
```

ユニットテストのプログラム

test_sample_unittest.py

```python
import unittest
import sample_unittest

class TestSample(unittest.TestCase):
    def test_add(self):
        actual = sample_unittest.add(1,2)
        expected = 3
        self.assertEqual(actual, expected)

if __name__ == "__main__":
    unittest.main()
```

ファイル構成

```
├── sample_unittest.py
└── tests
    └── test_sample_unittest.py
```

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

ユニットテスト 実行例

ユニットテストの実行(成功例)

```
python3 -m unittest tests.test_sample_unittest
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

ユニットテストの実行(失敗例)

```
python3 -m unittest tests.test_sample_unittest
F
======================================================================
FAIL: test_add (tests.test_sample_unittest.TestSample)
----------------------------------------------------------------------
Traceback (most recent call last):
  File ”xxx/tests/test_sample_unittest.py", line 12, in test_add
    self.assertEqual(expected, actual)
AssertionError: 3 != 4

----------------------------------------------------------------------
Ran 1 test in 0.000s
```

関数かテストケースのいずれかが
間違っている可能性あり

(※) 「python3 –m unittest」と入力することで全テストをまとめて実行することも可能です。
その場合、testディレクトリ内に空ファイル「__init__.py」を設置する必要があります。

その他の開発に関連する便利ツール

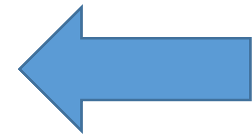
- Excelを操作するPythonライブラリ
  - OpenPyXL
- Webアプリケーションを開発するためのPythonフレームワーク
  - Django, Flask
- プログラムのバージョン管理システム
  - Git, GitHub, GitLab
- 継続的インテグレーションシステム
  - Jenkins, CircleCI
- 各ベンダの仮想ルータを利用できるサービス・方法
  - NetworkToCode On Demand Labs
  - Cisco CML, VIRL
  - ローカル環境に仮想ルータを構築する方法(個人ブログ)
    - Vagrantでfireflyを動かしたら自動化開発が捗った話
    - IOS-XRv Vagrantを試してみた

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY

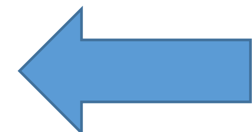
本日のまとめ

- 発表のねらい
- Python 基礎編
  - print
  - 変数 (数値型、文字列型、リスト型、辞書型)
  - if
  - for
  - ファイル入出力
  - 関数
  - クラス
- Python 応用編
  - ライブラリ
  - JSON
  - テンプレートエンジン
  - 正規表現
  - ユニットテスト
- ネットワーク自動化開発に便利なOSSライブラリ/ツールの紹介
- やってみようネットワーク自動化
  - プログラムからルータにログイン、showコマンドを実行
  - プログラムからルータにコンフィグ投入
- 自動化サンプル: BGP Peering作業を自動化する

事前オンライン発表
の範囲
(今回の対象)



JANOG41
の範囲
(今回は対象外)



Challenging Tomorrow's Changes

The corporate logo mark embodies our burning vision “to target more than swift perception of global changes and proper response to market shifts -- aspiring to be a part of inducing those transitions.” Beneath the logo, this desire is expressed in the phrase “Challenging Tomorrow's Changes.”

Copyright © 2018 ITOCHU Techno-Solutions America, Inc. CONFIDENTIAL ONLY