Bulletin of Aichi Univ. of Education, 68 (Natural Sciences), pp. 15-22, March, 2019

高等学校情報からの中学校技術分野のプログラミング言語の提案

安本　太一

情報教育講座

A Proposal for a Programming Language in Junior High School Technology Education from High School Informatics

Taichi YASUMOTO

Department of Information Sciences, Aichi University of Education, Kariya 448-8542, Japan

1　はじめに

新しい学習指導要領が，中学校では 2021 年度から完全実施，高等学校では 2022 年度から年次進行で実施される．中学校技術・家庭科技術分野 (以下中学校技術という) では，計測・制御のプログラミングに加え，新たに，ネットワークを利用した双方向性のあるコンテンツのプログラミングを取り上げ，情報セキュリティについても充実することになった [1]．高等学校では情報 I(以下高校情報という) の新設によりプログラミング，ネットワーク (情報セキュリティを含む) やデータベース (データ活用) の基礎が必修となっている [2]．

授業時間数は変わらないのに，中学校と高等学校ともに，プログラミングに関連する内容が増えている．このような状況の中で，高校情報の立場から，中学校技術でどのようなプログラミング言語を選定すべきなのかを提案する．

2　高校情報で想定されるプログラミングとプログラミング言語

生徒に，プログラミング，ネットワーク，データベースを関連づけて総合的に学んでもらうために，Web ベースのアンケートシステムを構築してもらうことが考えられる [2]．アンケートの入力受付と集計結果表示を，Web ベースで行うシステムを構築するのである．具体的には，プログラムをホームページ (Web サーバ) 上で動かし，プログラムからネットワーク経由でデータベースにアクセスするようにする．プログラムを記述するプログラミング言語は，Python が良いと思われる．Python は対話的な実行が可能でプログラムの構造が分かり易く可読性が高いといった特徴に加え，Python 自身が簡易 Web サーバと SQLite データベースの機能を備えているからである．Web サーバ，データベースを，プログラムの実行環境と別に用意した方が良いが，なんらかの事情で用意できない場合は，Python だけで完結することができる．Python には CGI をサポートするためのモジュール cgi がある．データーベースが PostgreSQL なら，モジュール psycopg2 を利用して，Python のプログラムがデータベースにアクセスできる [3]．

Python はシミュレーションや統計解析のためのモジュールが充実しているので，高校情報のシミュレーションや統計解析に Python を使うことができる [4]．Python なら，プログラミング，シミュレーション，統計解析をワンストップで行うことができることからも，高校情報では Python を使うのが良いと思われる．

3　中学校技術と高校情報のプログラミングの連続の必要性

現在の中学校技術におけるプログラミングは，教材メーカから販売されている個人持ち教材 (ロボットなど) 専用ソフトウェアを用いたビジュアルプログラミングや，ドリトルといった日本語プログラミングが行われていることが多い [5] ．このようになっているのは，過去の授業で使われた BASIC 言語が，英語表記でかつプログラムの構造が分かりにくいため，授業がうまくいかなかったことが理由であると推測される．しかしながら，教材専用のプログラミング環境や日本語プログラミングは理解しやすいという点では優れているが，世間や海外で使われているプログラミング言語とは異なるという観点からみると，実社会との接点がなく中学校技術の授業までにとどまってしまい，その先の発展がないのが残念である．

近年，学習指導要領の改定や社会の状況の変化があったことから，次のようなことが考えられる．

- 今後は，高等学校でプログラミングが必修になるので，中学校技術と高校情報で，プログラミング言語を統一した方が，中高におけるプログラミングの授業の連続性ができ，生徒にメリットがある．

- 学習指導要領の改訂により，小学校で英語と (プログラミングとともに) 基本的な文字入力が必修となっているので，かつて BASIC 言語を使ってプログラミングの授業が行われた時に比べれば，英語表記やタイピングへの抵抗感は小さくなっていると考えられる．
- 中学校技術ではネットワークプログラミングが必修となり，従来よりも多くのことを生徒に学ばせることになるが，教育現場で閉じている教材専用のプログラミング環境や日本語プログラミングを用いて，ネットワークプログラミングまで引っ張って良いのかという疑問がある．実際のところ，micro:bit という英国で開発された小学生向けのボードコンピュータのプログラミング環境では，ビジュアルプログラミングに加えて，JavaScript，Python の環境が用意されているからである．
- 現在，日本国内でも，Python の普及が進んでおり，Python を扱った書籍や雑誌が数多くある．

以上のことから，中学校技術の段階から，世界共通のテキストベースプログラミング言語の一つである Python を導入したら良いのではないかという考えに至った．そこで，新学習指導要領における中学校技術のプログラミングを Python で行うとどのようになるか，検討してみる．

4 双方向性のあるプログラミングとしてのチャットプログラム

ネットワークを利用した双方向性のあるプログラミングとして，第一にあげられるのは，チャットのプログラミングであろう [1]．Python で記述した典型的なチャットのサーバプログラムを図 1 に，クライアントプログラムを図 2 に示す．同一のコンピュータで，容易に試せるようにサーバの IP アドレス`server_ip` はループバックアドレスになっているが，それぞれのプログラムを異なるコンピュータで実行する場合は`server_ip` をサーバプログラムを動かすコンピュータの IP アドレスに変更する必要がある．

これらのプログラムの流れは，書籍や雑誌に掲載されている UNIX のネットワークプログラミングの C 言語版チャットプログラムの典型的な例題とほぼ同じである [6]．Python 版は，C 言語版に比べると，はるかに少ない文字数・行数で収まっている．その理由は，次のとおりである．

- Python の変数はどの型のデータも代入でき，事前に変数宣言をする必要がない．C 言語においてみられる `struct sockaddr_in 変数名` というよう

```
# -*- coding:utf-8 -*-
import socket

server_ip = '127.0.0.1' # サーバの IP アドレス
server_port = 50000 # サーバがクライアントから接続を受けるポート番号

# サーバ用のソケットを作成し，server_socket に格納
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # すぐ再実行に必要
server_socket.bind((server_ip,server_port)) # サーバの IP アドレスとポート番号を指定

# 待ち行列の長さ 1 で，クライアントからの接続を受付
print('チャットプログラム ( サーバ版 ) 開始．クライアントからの接続要求の受付を始めます．')
server_socket.listen(1) # 接続要求の受付を始める ( 接続の要求を聞き始める )

# 接続要求を 1 つ受理する．
client_socket, client_ip = server_socket.accept() # 接続を受理すればこの行を通過
print('クライアントからの接続を 1 つ受け付けました．')

# メインループ：クライアントからメッセージの受信，クライアントへのメッセージ送信の繰り返し
while True:
    received_message = client_socket.recv(1024) #1024 はメッセージ最大長 (2 のべき乗 )
    print('クライアントから受信しました (Enter だけなら終了 ):'+ received_message.decode())
    if len(received_message) == 0:
        break

    print ('送信メッセージを入力してください (Enter だけなら終了 ):')
    send_message = input()
    client_socket.sendall(send_message.encode()) # クライアントへメッセージ送信
    if len(send_message) == 0:
        break

    print ('クライアントへメッセージを送信しました．クライアントからメッセージ待ち ')

# 切断処理
client_socket.close()
server_socket.close()
```

図 1: チャットのサーバプログラム平文版

```
# -*- coding:utf-8 -*-
import socket

server_ip = '127.0.0.1' # サーバーの IP アドレス
server_port = 50000 # サーバのポート番号

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # ソケットの作成をする

print('チャットプログラム ( クライアント版 ) 開始．クライアントへ接続します ...')
client_socket.connect((server_ip, server_port)) # サーバーに接続

print('サーバに接続しました．送信メッセージを入力してください (Enter だけなら終了 ):')
send_message = input()

# メインループ：サーバへメッセージを送信し，サーバからのメッセージの受信の繰り返し
while True:
    client_socket.send(send_message.encode()) # サーバへメッセージ送信
    if len(send_message) == 0:
        break
    print ('サーバへメッセージを送信しました．サーバからのメッセージ待ち ')
    received_message = client_socket.recv(1024) # 数字は受信バッファの大きさ (2 のべき乗 )
    print('サーバから受信しました (Enter だけなら終了 ):'+ received_message.decode())
    if len(received_message) == 0:
      break
    print ('送信メッセージを入力してください (Enter だけなら終了 ):')
    send_message = input()

# 切断処理
client_socket.close()
```

図 2: チャットのクライアントプログラム平文版

な，ネットワークプログラミングをするための複合データ (構造体) の名前を伴った変数宣言が不要である．

- `#include <sys/socket.h>`といったC言語においてみられる数行に及ぶネットワークプログラミングのためのヘッダインクルードが，`import socket`という1行のモジュールのインポートで済んでいる．標準入出力や文字列のためのヘッダインクルードに対応するモジュールのインポートもない．
- エラー処理は，Pythonの実行環境の標準の例外処理にまかせることにして，明示的に記述していない．

これでも，中学生には難しいという意見もあるだろうが，例えば図1の8〜10, 14, 17, 36, 37行目や図2の7, 10, 29行目のようにソケットプログラミングに定型な部分が多くあり，プログラミングの熟練者でも定型な部分は当然のように入力していることが多い．定型な部分は“こういうもの”ととらえ，教員や生徒が難しく考える必要はない．ただし，何も説明しないのは乱暴なので，大雑把に説明できることは，説明する必要がある．例えば，ネットワークプログラミングのソケット (socket) は，物理的なコードの端についている部品のようなもので，IPアドレスやポート番号が紐付けされているといった説明をプログラムの該当部分 (図1の10行目や図2の10行目) を指しながら行う必要がある．

```
 1:# -*- coding:utf-8 -*-
 2:import socket, ssl
 3:
 4:server_ip = '127.0.0.1' # サーバの IP アドレス
 5:server_port = 50000 # サーバがクライアントから接続を受けるポート番号
 6:
 7:# サーバ用のソケットを作成し，server_socket に格納
 8:server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 9:server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # すぐ再実行に必要
10:server_socket.bind((server_ip,server_port)) # サーバの IP アドレスとポート番号を指定
11:
12:# 暗号化通信用の包装紙のようなもの ( 中身を隠すもの ) を作成
13:context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
14:context.load_cert_chain(certfile="keys/server.crt", keyfile="keys/server.key")
15:
16:# 待ち行列の長さ 1 で，クライアントからの接続を受付
17:server_socket.listen(1) # 接続の待ち受けを開始
18:print(' チャットプログラム ( サーバ版 ) 開始．クライアントからの接続を待ちます ...')
19:
20:# 接続要求を 1 つ受理する．
21:client_socket, client_ip = server_socket.accept() # 接続を受けつければこの行を通過
22:print(' クライアントからの接続を 1 つ受け付けました．')
23:
24:connstream = context.wrap_socket(client_socket, server_side=True)# 包装紙をかぶせる
25:print(' 暗号化通信を開始しました．')
26:
27:# メインループ：クライアントからメッセージの受信，クライアントへのメッセージ送信の繰り返し
28:while True:
29:    received_message = connstream.recv(1024)#1024 はメッセージ最大長 (2 のべき乗 )
30:    print(' クライアントから受信しました (Enter だけなら終了 ):'+ received_message.decode())
31:    if len(received_message) == 0:
32:        break
33:
34:    print (' 送信メッセージを入力してください (Enter だけなら終了 ):')
35:    send_message = input()
36:    connstream.sendall(send_message.encode()) # クライアントへメッセージ送信
37:    if len(send_message) == 0:
38:        break
39:
40:    print (' クライアントへメッセージを送信しました．クライアントからメッセージ待ち ')
41:
42:# 切断処理
43:connstream.shutdown(socket.SHUT_RDWR)
44:connstream.close()
45:server_socket.close()
```

図 3: チャットのサーバプログラム暗号化版

5 チャットプログラムの情報セキュリティの充実

新学習指導要領では，情報セキュリティについても充実するとなっているので，暗号化通信を取り上げる．

5.1 平文版プログラムのSSL暗号化

チャットプログラムが安全に利用できるための方策として，通信の暗号化があげられる．Pythonでは，図1，図2のプログラムを，業界標準の暗号化方式SSLを用いて，それぞれ，図3，図4のように書き換えることができる．暗号化のために追加したり，書き換えたりした部分を，赤字で示している．OpenSSLで生成したSSL電子証明書などは，keysというディレクトリ (フォルダ) を作成して，その中に置いている．自己署名証明書を使うことになってしまうだろうが，ここでは暗号化通信を実現してその効果を確認できれば良いので， 構わないことにする．暗号化版は，平文版と大枠は同じだが，平文版のソケット`client_socket`を，通信の内容がみられないように，包装紙 (wrapper) をかぶせているような雰囲気が見て取れる．

```
 1:# -*- coding:utf-8 -*-
 2:import socket, ssl
 3:
 4:server_ip = '127.0.0.1' # サーバーの IP アドレス
 5:server_port = 50000 # サーバのポート番号
 6:
 7:client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # ソケットの作成をする
 8:
 9:# 暗号化通信用の包装紙のようなもの ( 中身を隠すもの ) を作成
10:context = ssl.create_default_context()
11:context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
12:context.verify_mode = ssl.CERT_NONE
13:context.check_hostname = False
14:
15:# 接続用のソケットに包装紙をかぶせる
16:connstream  =  context.wrap_socket(client_socket, server_hostname=server_ip)
17:
18:print(' チャットプログラム ( クライアント版 ) 開始．クライアントへ接続します ...')
19:connstream.connect((server_ip, server_port)) # サーバーに接続
20:
21:print(' クライアントに接続しました．送信メッセージを入力してください (Enter だけなら終了 ):')
22:send_message = input()
23:
24:# メインループ：サーバへメッセージを送信し，サーバからのメッセージの受信の繰り返し
25:while True:
26:    connstream.sendall(send_message.encode()) # サーバへメッセージ送信
27:    if len(send_message) == 0:
28:        break
29:
30:    print (' サーバへメッセージを送信しました．サーバからのメッセージ待ち ')
31:    received_message = connstream.recv(1024) #1024 はメッセージ最大長 (2 のべき乗 )
32:    print(' サーバから受信しました (Enter だけなら終了 ):'+ received_message.decode())
33:    if len(received_message) == 0:
34:      break
35:
36:    print (' 送信メッセージを入力してください (Enter だけなら終了 ):')
37:    send_message = input()
38:
39:# 切断処理
40:connstream.shutdown(socket.SHUT_RDWR)
41:connstream.close()
```

図 4: チャットのクライアントプログラム暗号化版

5.2 暗号化の効果の確認

暗号化の効果の確認は，ポートミラーリング機能を有するスイッチングハブをサーバとクライアントの間の経路のどこかに設置し (挿入し)，そのスイッチングハブのミラーポートからでてくるサーバとクライアントの間のパケットを，Wireshark などのネットワークアナライザソフトウェアで観察することであろう．

図 5 に，平文版である図 1 と図 2 のプログラムが通信して，はじめにクライアントからサーバへ “My name is Hanako.”，次にサーバからクライアントへ “My name is Taro.” というメッセージを送って，終了した時のパケットを Wireshark で観察している様子を示す．Raspberry Pi というボードコンピュータ 2 台で，サーバプログラム，クライアントプログラムを個別に動作させている．Windows パソコンでもこれらのプログラムは動作するが，筆者のコンピュータ環境の都合で，ボードコンピュータで実行している．サーバの IP アドレスは 10.0.1.24，クライアントの IP アドレスは 10.0.1.25 である．4 番目のパケットがクライアントからサーバへのメッセージ，6 番目のパケットがサーバからクライアントへのメッセージである．図 5 では，4 番目のパケットが選択されて，アプリケーション層までの詳細が表示されている．“My name is Hanako.” と表示されていて，平文なら盗聴できてしまうことが容易にわかる．セキュリティの充実といって，パスワード認証を導入しても，平文ならパスワードが盗聴されてしまうことが容易に実感できるであろう．

図 6 に，暗号化版である図 3 と図 4 のプログラムが通信して，はじめにクライアントからサーバへ “My name is Hanako.”，次にサーバからクライアントへ “My name is Taro.” というメッセージを送って，終了した時のパケットを Wireshark で観察している様子を示す．12 番目のパケットがクライアントからサーバへのメッセージ，14 番目のパケットがサーバからクライアントへのメッセージである．図 6 では，12 番目のパケットが選択されて，アプリケーション層までの詳細が表示されている．アプリケーション層には，“My name is Hanako.” とは全く無関係に見えるデータが出現していて，暗号化の効果が確認できる．平文版ではキーボード入力された文字の数 18 バイトだったのが，暗号化版では 42 バイトとバイト数が増えている．暗号化すると，データ量が増える理由を，中学生に直感的に考えてもらったり，教員が直感的に説明するのが良いだろう．

5.3 中学校技術と高校情報におけるパケット観察の位置付け

図 5 の平文版の 10 個のパケットを全部いきなりみせられても，生徒は困惑してしまう．そこで，Wireshark でリアルタイムでパケットを採取・表示するようにしておき，チャットプログラムの実行時に，接続の確立で (すぐチャットのためのキーボード入力に移行せずに) 一休止，チャット即ち正味の通信で (すぐ返答をせずに) 一休止，終了のための Enter だけの入力も一休止してから行えば，表示がその都度停止するので，10 個あるパケットの内訳 (接続の確立，クライアントからサーバへの正味の通信，サーバからクライアントへの正味の通信，接続の切断) が TCP の伝送制御のことを知らなくても体感的に理解できるであろう．このようにして，正味の通信の範囲のパケットを特定して，キーボード入力の内容が盗聴できてしまうことを確認するのである．

図 6 の場合も同様であるが，図 3 の 24 行目の実行の前に`time.sleep(停止秒数)` を一時的に挿入 (冒頭で time モジュールの import も必要) すれば，18 個あるパケットの内訳 (接続の確立，暗号化通信の準備，クライアントからサーバへの正味の通信，サーバからクライアントへの正味の通信，接続の切断) が体感的に理解できる．そして，正味の通信の範囲のパケットを特定しても，キーボード入力の内容が表れていない (わからない) ことで，暗号化通信の効果を確認するのである．

中学校技術では，プロトコル階層は取り扱わず，正味の通信の範囲のアプリケーション層 (つまり図 5 や図 6 の一番下) だけをみて，暗号化の効果だけを理解できれば良いだろう．

一方，高校情報では，伝送制御やプロトコルの階層構造まで扱うことになっているので，接続の確立の`[SYN]`，`[SYN, ACK]`，`[ACK]` のパターン，キー入力のデータに対する確認応答の`[ACK]`，接続の切断の`[FIN, ACK]`，`[FIN, ACK]`，`[ACK]` のパターン，これらの TCP フラグの場所およびその意味の理解が必要である．Wireshark の表示の三角形▲がついているところを見たりクリックして，各層のヘッダや，上下の層の関係について理解することも必要である．

6 計測・制御のプログラミング

新学習指導要領では，中学校技術において，計測・制御のプログラミングも引き続き行うことになっているので，Python による計測・制御のプログラミングの例を示す．

図 7 に，人を検知したらブザーを鳴らすプログラムを示す．Raspberry Pi というボードコンピュータの GPIO17 にブザーを，GPIO27 に人感センサー (赤外線センサー) を接続している状態で，人を感知したらブザーを鳴らすプログラムを示す．10 行目までで GPIO のセットアップをしている．13 行目からのループでは，人感センサーが人を感知したら (GPIO27 がデジタル 2 値の 1 になったら) ブザーを鳴らし (GPIO17 をデジタ

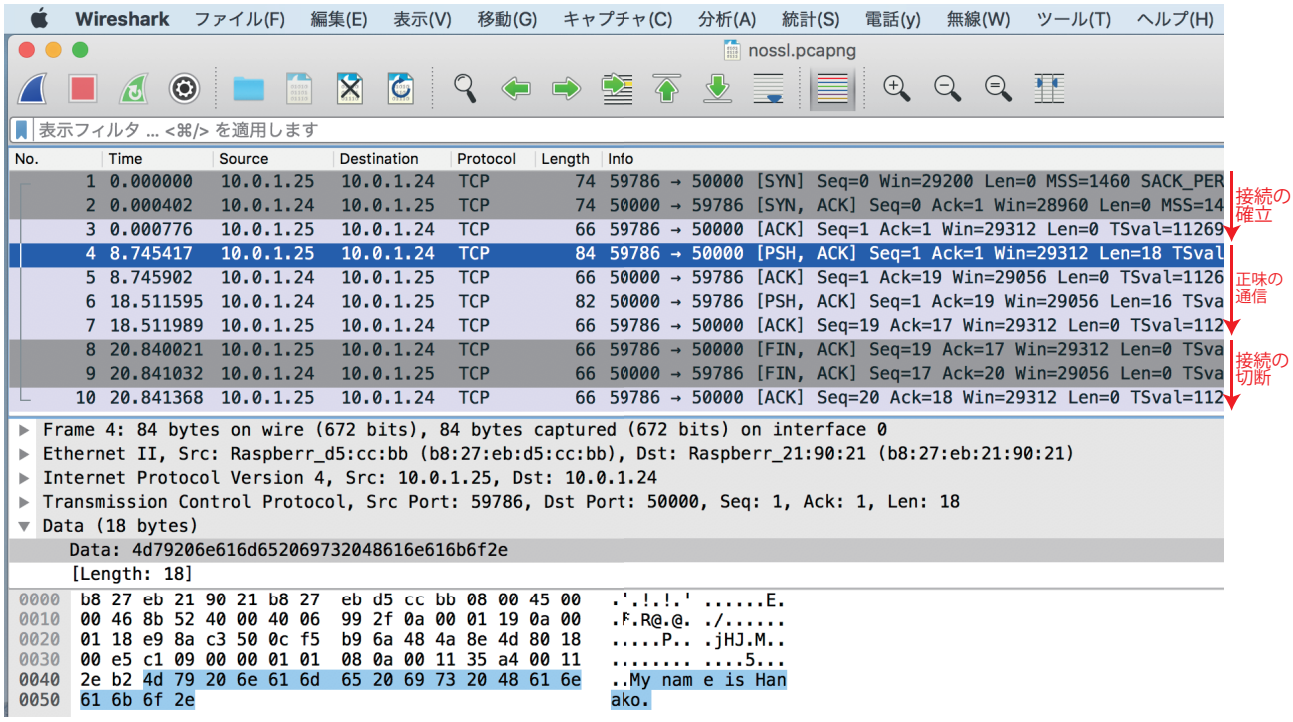

図 5: 平文版のパケットの観察

ル 2 値の 1，すなわち 3.3V にし)，さもなければブザーを鳴らさない (GPIO17 をデジタル 2 値の 0，すなわち 0V にする) ということを繰り返している．

```
 1:# -*- coding: utf-8 -*-
 2:#GPIO を使うためのモジュールをインポート
 3:import RPi.GPIO as GPIO
 4:
 5:# 入出力のモードをセット
 6:GPIO.setmode(GPIO.BCM) # デバイスの番号を GPIO 番号にする
 7:buzzer = 17 # ブザーの番号
 8:sensor = 27 # 人感 ( 赤外線 ) センサーの番号
 9:GPIO.setup(buzzer, GPIO.OUT) # ブザーは電圧の出力
10:GPIO.setup(sensor, GPIO.IN)  # センサーは電圧の入力
11:
12:# メインループ
13:while True:
14:    if GPIO.input(sensor): # センサーが人 ( 赤外線 ) を感じたら
15:        GPIO.output(buzzer, True) # ブザーを鳴らす
16:    else:
17:        GPIO.output(buzzer, False) # ブザーを鳴らさない
```

図 7: 人感センサー付ブザーのプログラム

図 7 は入力と出力が共にデジタル 2 値であったので，入力が多値を扱う例として図 8 に気温が 28 ℃以上になったらプロペラがついたモータを回すプログラムを示す．Raspberry Pi というボードコンピュータの I2C インタフェースに ADT7410 という温度センサーを，GPIO17 にモータドライバ (FET あるいは専用 IC であるモータドライバの先のモータにはプロペラがついている) を接続している．図 8 のプログラムは， 図 9 のプログラム (モジュール) をインポートしている．

一般に，多値の場合は，センサーの出力値をそのまま使えることはなく，温度などの我々の生活に使っている値にするための換算 (計算) が必要である．換算は，図 9 の 11〜14 行目にみられるように，四則演算のほか，ビット演算を伴うことがあり，中学生には難易度が高いかもしれない．多値を扱うセンサーをコンピュータに接続するインタフェースの説明も，中学生には難易度が高いかもしれない．そのような場合は，Python のモジュール機能を用いて，換算やインタフェースの詳細を隠すことができる．図 8 では，adt7410 というモジュールから temp という温度を摂氏で返す関数をインポートしている．教員が，あらかじめ，多値のセンサーを使うためのモジュールを提供しておいて，生徒にはそのモジュールを使わせるという授業展開が考えられる．全てを知らなくてもプログラムが書けることや，またそのような仕組みがあってそれがプログラミング言語の重要な機能であることを，生徒に説明する機会も得られる．計測・制御のプログラムの方が，ネットワークのチャットのプログラムより，行数が少ないので，生徒が難易度が低いと感じるかもしれない．

a

7 デバッグ

新学習指導要領においても，中学校技術のプログラミングでは，デバッグができることと述べられている．

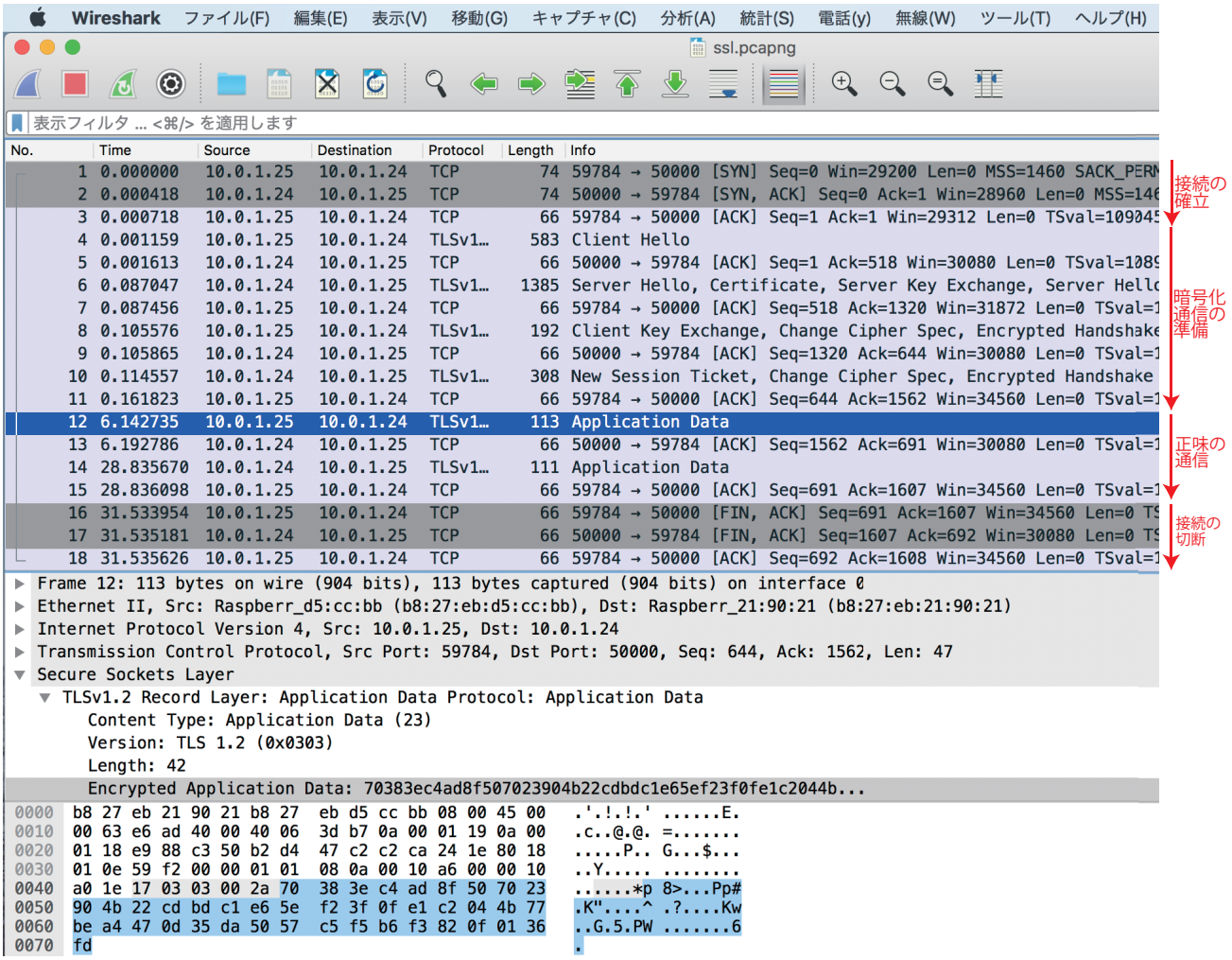

図 6: 暗号化版のパケットの観察

```
 1:# -*- coding: utf-8 -*-
 2:import RPi.GPIO as GPIO #GPIO のためのモジュールをインポート
 3:from adt7410 import temp # 温度センサーのモジュールをインポート
 4:
 5:# 入出力のモードをセット
 6:GPIO.setmode(GPIO.BCM) # デバイスの番号を GPIO 番号にする
 7:motor = 17 # モータドライバの番号
 8:GPIO.setup(motor, GPIO.OUT) # モータドライバへは電圧の出力
 9:
10:# メインループ
11:while True:
12:    if temp() >= 28.0 : #28℃以上なら
13:        GPIO.output(motor, True) # モータを回す
14:    else:
15:        GPIO.output(motor, False) # モータを回さない
```

図 8: 温度センサー付き扇風機のプログラム

```
 1:# -*- coding: utf-8 -*-
 2:#I2C インタフェース接続の ADT7410 から温度を読み取る
 3:import smbus
 4:
 5:i2c = smbus.SMBus(1)
 6:address = 0x48
 7:
 8:# 温度を摂氏で返す
 9:def temp():
10:    block = i2c.read_i2c_block_data(address, 0x00, 12)
11:    t = (block[0] << 8 | block[1]) >> 3
12:    if(t >= 4096):
13:        t -= 8192
14:    t = t / 16.0
15:    return t
```

図 9: 温度センサーのプログラム (adt7410.py)

Python では，テキストプログラミングにおいて伝統的な出力文 (例えば print) の挿入によるデバッグが可能である．プログラムの実行状況や実行場所などを示すメッセージや変数の値を表示したり，それらの履歴をテキストとして残すことができる．メッセージや変数の履歴をテキストとして残すことは，ビジュアルプログラミングでは，一般に難しいと考えられる．

通常インストールされる統合開発環境 Python IDLE では，一般的なソースコード・デバッガの機能を使うことができる．プログラムのステップ実行の機能があって，ソースコードにおける制御の流れや，変数などの内容を確認できる．デバックをしない時でも，プログラムの理解に役に立つ機能である．

新学習指導要領において高校情報のプログラミングではデバッグについて言及されていないことから，中学校技術のプログラミングにおいて，出力文の挿入やデバッガによるデバッグが取り扱われることが望ましい．

8 まとめ

高校情報の立場から，中学校技術におけるプログラミング言語として Python を提案し，Python によるネットワークを利用した双方向性のあるコンテンツのプログラミングや計測・制御のプログラミングの例を示した．計測・制御のプログラムは理解が難しくなるほど長いものではなく，ネットワークのプログラムは長いが定型の部分が多いだけである．計測・制御の方が，プログラムが短いことから，学習指導要領に掲げられている順とは異なるが，計測・制御から扱う方が望ましい．

文部科学省の委託授業による諸外国におけるプログラミング教育に関する調査研究 [7] によると，英国 (イングランド) では，7 年生で Kodu，8 年生で Scratch，9 年生で Scratch と Python，韓国では中学校で Scratch，高等学校で Python ，フィンランドでは 3–6 年生ではビジュアルプログラミング，7–9 年生ではプログラミング言語を学び始めるという報告がある．このように，諸外国では，テキストベースのプログラミング言語の導入は，中学校あるいは高校で行われている．Python は，上記の国以外の報告でも名前がでている．

テキストベースのプログラミング言語の導入は，生徒と教員にとって，一つの大きな壁である．中学校技術が，生活や社会との接点を重要視するならば，実社会で使われているテキストベースのプログラミング言語を採用するのは方策としてあり得る．高等学校進学者が多いとはいえ，義務教育が中学校で終わるからである．

中学校技術，高校情報の全ての授業時間を，プログラミングに使えるわけではない．高校情報のプログラミングはテキストベースのプログラミングが必要である．大学でプログラミングを教えた経験から，高等学校からテキストベースのプログラミング言語を扱ったのでは，十分な時間が確保できず，プログラミングが身につかない者が多く出ることは明らかである．そこで，中学校と高等学校の 2 つの学校種にまたがって，同一のテキストベースのプログラミング言語に取り組むことが，現実的な対応と考えられる．

今後は，新学習指導要領実施までに，公開講座などの教員向け研修や，学校への訪問授業で，Python によるプログラミングを扱い，教員や生徒の受け止め方を探る予定である．

参考文献

[1] 文部科学省：中学校学習指導要領解説 技術・家庭編, (2017).

[2] 文部科学省： 高等学校学習指導要領解説 情報編, (2018).

[3] Gregorio, F. D. and Varrazzo, D. : Psycopg - PostgreSQL database adapter for Python, http://initd.org/psycopg/docs/.

[4] Shaha, A. : *Doing Math with Python : Use Programming to Explore Algebra, Statistics, Calculus, and More!*, No Statch Press (2015).

[5] 井戸坂幸男：プログラミング入門をどうするか：4. 中学校におけるプログラミング教育 - 制御プログラムとソフトウェアの仕組み理解を中心として-, 情報処理, Vol.57, No.4, pp.354–357 (2016).

[6] 雪田修一：UNIX ネットワークプログラミング入門, 技術評論社 (2003).

[7] 大日本印刷株式会社：「諸外国におけるプログラミング教育に関する調査研究」文部科学省平成 26 年度・情報教育指導力向上支援事業 (2015).

（2018 年 9 月 25 日受理）