2016年度 卒業論文

ウェアラブル機器を用いた脈拍の観察及び異常通知の提案

大阪産業大学 工学部 情報システム工学科
情報教育システム研究室

13H070 西村龍介

目次

1　はじめに

2016 年版高齢社会白書 [1] では総人口 (1 億 2,711 万人) に占める 65 歳以上人口の割合 (高齢化率) は 26.7%(3,393 万人) である。4 人に 1 人が 65 歳以上になっている状況で今後も増え続るであろう。又、年々医療費も増加している傾向にある [2]。このような状況では病気の早期発見・治療ができれば医療費を抑えることができると考えた。そこでウェアラブル機器 (Charge HR) を使用して橈骨動脈の脈拍を測定し、データの分析や観察を行うことで病気の通知や予防をする。その中でも心臓突然死の 70% を占め不整脈について研究を進める。又、不整脈の 1 種である心房細動患者数は、検診で診断される患者数だけでも約 80 万人と推計されている [3]。この病気は高齢者に多く、今後人口の高齢化に伴ってさらに増加すると予想され、健康な人にもしばしば起きているから軽視できないと考えた。

第 2 章では本研究の目的について述べる。第 3 章では本研究で開発したシステムの概要について述べる。第 4 章では本研究で使用したハードウェアについて述べる。第 5 章ではシステムの結果と考察について述べる。第 6 章には研究の成果についてまとめる。

2　目的

不整脈は実は必ずしも心臓が悪いから起こるものではない。不整脈の原因として最も多いのは年齢に伴うものや、体質的なものである。中年以上ではほとんどの人に、毎日 1~2 個は不整脈が見つかる。歳をとるにつれ、だれでも少しずつ不整脈が増え、ストレス、睡眠不足、疲労などでも不整脈は起こりやすくなる。そういう意味では、だれにでも起こりうるものだと言える。

本研究では、日々の脈拍データを分析・観察することで不整脈だけでなく脈拍で判る持病の早期発見・予防を視野に早期治療ができることを目的とする。

3　システム

本章では、不整脈、従来の不整脈診断方法と本システムの方法について解説する。

3.1　不整脈

不整脈は、脈の打ち方がおかしくなることを意味する。この中には異常に速い脈（頻脈）や遅い脈（徐脈）も含まれる。「脈」とは、心臓から押し出される血液の拍動が血管に伝わって感じられるものである。もし心臓のリズムに異常が起きれば、脈は乱れる。心臓は筋肉でできた臓器で、その筋肉にかすかな電気が流れて興奮し、動く仕組みになっている。

心臓でどう電気が伝わっていくかを図 4 に示す [5]。心臓の上の方にある「洞結節」というところで電気がつくられ、電気の通り道（伝導路）を通って心臓全体に流れ、筋肉が収縮するようになっている。

例えば、洞結節で電気が発生しない、または別の場所から電気が流れてしまうと、心臓が規則正しく興奮しなくなる。つまり、不整脈は心臓に流れる電気の異常や刺激が伝導路をうまく伝わらないことを意味する。

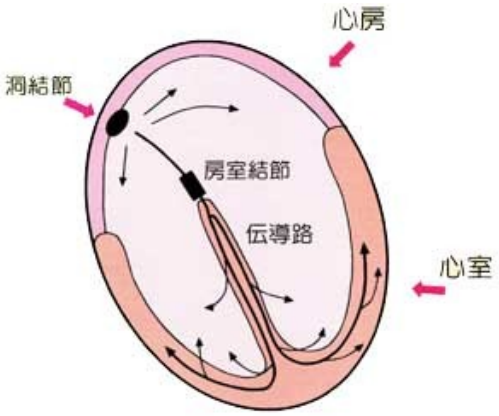

図 1　心臓での電気の伝わり方。国立循環器病研究センター循環器病情報サービス [5] より引用

3.1.1　不整脈の種類

不整脈には大きく分けると 3 種類がある。不整脈の 3 種類を表 1 に示す。

表 1　不整脈の種類

徐脈	洞不全症候群
	房室ブロック
頻脈	心室頻拍
	心房細動
	心室頻拍
	発作性上室性頻拍
	WPW 症候群
期外収縮	心房性期外収縮
	心室性期外収縮

・徐脈

心臓を動かすのに必要な電気がつくられなくなったり、途中で途切れてしまったりして脈が異常に遅くなるのが徐脈である。脈が遅くなるとだるさを感じ、体を動かすのがつらくなる。動作をするたびに激しい息切れが生じることもある。1 分間に 40 回くらいまで脈拍が減ったときは危険な状態である。

・頻脈

運動や緊張とは関係なく、脈が異常に速くなるのが頻脈である。脈拍が増えると動悸が起こり、ひどい場合にはめまいや冷や汗、吐き気を生じる。意識を失うこともあるので注意が必要です。

頻脈は、心臓に送られる電気が異常に早くつくられることが原因であるとされている。また、電気の通り道に異常が生じて電気信号が本来の働きをしていない可能性も考えられる。脈が突然 1 分間に 150 回以上になった場合は、危険な状態であると考えらる。

・期外収縮

期外収縮とは、脈が一時的にとんだり、不規則なリズムになったりする不整脈である。心臓を動かしている電気の刺激が通常とは違う場所から出てくるために脈が乱れる。その場所が心房の場合は心房性期外収縮、心室の場合は心室性期外収縮と呼ばれている。血圧の波は安定しているのが正常な状態ですが、期外収縮の場合には思わぬ収縮が起きるために波が乱れてしまう。

期外収縮は若い健康な方でも 1 日に何回か起こることがあり、直接治療が必要なものではない。しかし、連続して起こったり自覚症状がある場合には対応を考える必要がある。

3.2 従来の不整脈診断方法

・12 誘導心電図検査

最も一般的で基本的な検査です。ベッド上で横になり手足と胸に電極を貼り付け、12 ヶ所の心電図を記録する [6]。12 誘導心電図検査を図 5 に示す。

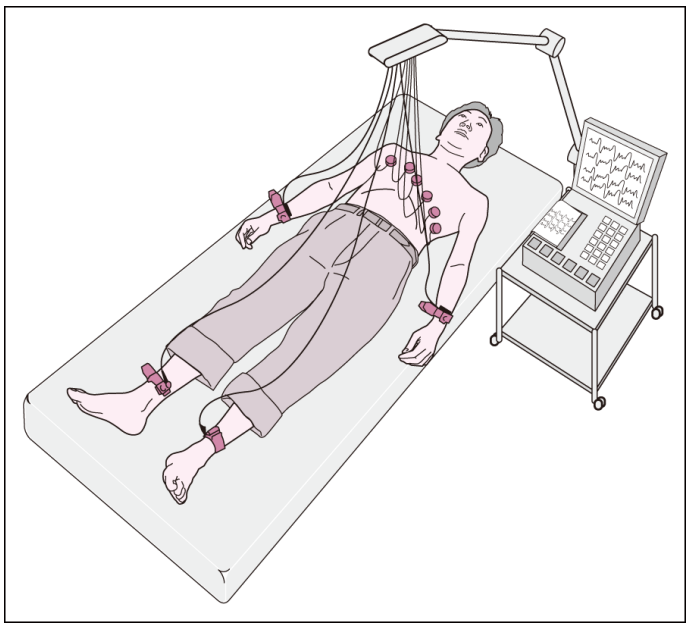

図 2　12 誘導心電図検査。看護 roo [7] より引用

・ホルター心電図（24 時間心電図検査）

医療機関で一時的に心電図検査をしても心臓の状態は正確には記録出来ない事がある。そこで患者さんの日常生活に合わせ、一日分の心電図をテープに記録し、より詳しく心臓の状態を解析するのが、ホルター心電図である [6]。

・運動負荷心電図（トレッドミル、エルゴメーター、マスター等）

安静時には見られない不整脈も運動時には悪化したり、逆に安静時に多発していた不整脈も運動時には解消する場合がある。それを検査室内で再現する為に、トレッドミル（＝ベルト上を走る）、自転車を漕ぐ（＝エルゴメーター）、階段昇降 (＝マスター) 等の運動負荷をかけながら心電図を記録する [6]。運動負荷心電図を図 6 に示す。

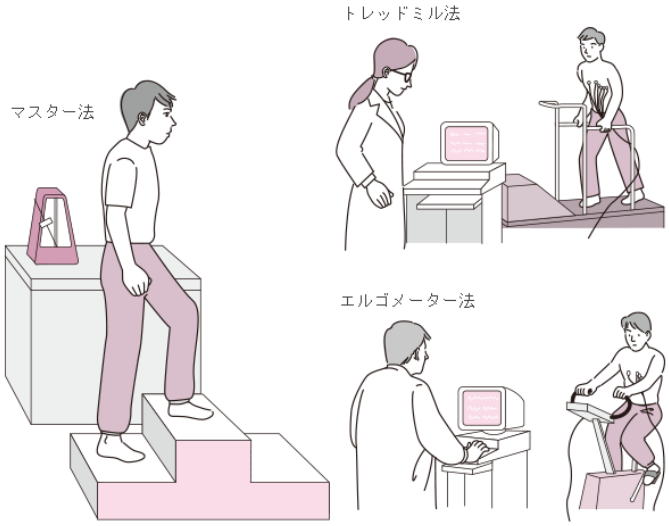

図 3　運動負荷心電図。Qlife・負荷心電図検査 [8] より引用

・心臓超音波検査（心エコー検査）

心臓病の診断やその重症度診断の為には、心臓弁や心臓筋肉の詳細な診断が必要になる。心エコー検査により、心臓の収縮する力、弁の動き、筋肉の厚さや動き、心臓の部屋（心房、心室）の大きさ等が検査出来るので、不整脈診断に必要な心臓自体の病気の有無が判る [6]。心臓超音波検査を図 7 に示す。

図 4　心臓超音波診断。QLife・心臓超音波検査 [9] より引用

・心臓電気生理検査（EPS 検査）

電極カテーテルという数ミリ径の細い管を、足の付け根や肩の下にある静脈から、心臓に向かって数本挿入する。このカテーテルの先端には金属製の小さなチップ（＝電極）が付いてある、これを心臓内壁に接触させると、心臓内の電気活動を詳細に得られる。この検査中に、色々な薬剤を投与したり、カテーテルを通じて心臓に電気刺激を与える事で、意図的に不整脈を起こして、患者さん特有の不整脈の原因、不整脈の発生元、重症度、有効性のある薬剤の判定等を行い、患者さんごとに最適な治療方針が決定出来る、不整脈診断に於いては非常に重要かつ有効な検査方法である。その他に、血液検査や胸部レントゲン検査も行う場合もある [6]。心臓電気生理検査を図 8 に示す。

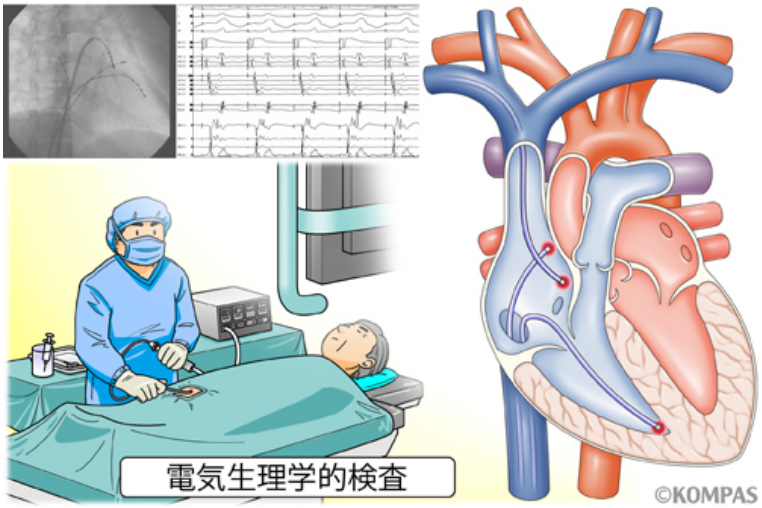

図 5　心臓電気生理検査。KOMPAS・心臓電気生理学的検査 [10] より引用

3.2.1　不整脈検査費用

表 2　不整脈検査費用。北岡クリニック・不整脈 [11]、筑波メディカルセンター病院・入院費用概算 [12] より引用

心臓の検査	検査料 (3 割負担)
心臓エコー	2,640 円
ホルター心電図 (24 時間)	4,500 円
運動負荷検査	960 円
心臓電気生理検査[*1](EPS) 左心	60,000 円
心臓電気生理検査 (EPS) 右心	90,000 円

※検査費用には、診察料や診療情報提供料などは含まれない

[*1] 一般的な医療機関でのカテーテルアブレーション治療はだいたい 2 泊 3 日の入院で行なわれている。2 泊 3 日のスケジュールで行える不整脈は、発作性上室性頻拍症（WPW 症候群、房室結節リエントリー性頻拍症、心房頻拍）、心房粗動、心室性期外収縮、（特発性）心室頻拍である。不整脈アブレーションは健康保険の適応があり、多くの生命保険・医療保険で手術給付金の対象となっている。入院治療費は、高額医療申請の手続きをすると約 10 万円の自己負担となる。

3.3 本システム

3.3.1 OAuth2

本システムは OAuth2 も用いて FitbitAPI を取得した。通常、オンラインのサービスを利用するためには、ログイン ID とパスワードを組み合わせた情報 (クレデンシャル) で認証するログイン認証になる。しかし、近年 Twitter と Facebook のような異なる Web サービスの連携が進み、デジタルアイデンティティの共有が問題となった。例を挙げると Twitter にユーザーの個人情報があるとき、Twitter と Facebook が連携し、Twitter にある個人情報を Facebook が自由にアクセスできる状況は好ましくない。そのため、Web サービスへのアクセスを認可する手段が必要とされていた。

例えば、悪意のあるアプリケーションが、ユーザに対してクレデンシャルを要求した場合、ユーザの個人情報が漏洩する危険がある。OAuth を使うと、他の Web サービスと連携するためユーザにクレデンシャルを譲り渡すという要求をしなくても、ユーザデータにセキュアにアクセスできるようになる。OAuth はリソースオーナーの代わりに保護リソースにアクセスする方法をクライアントに提供する。クライアントが保護リソースにアクセスする前に、クライアントはまずリソースオーナーの認可を取得し、アクセス許可とアクセストークン (権限の範囲と期間を示すトークン) を交換しなけらばならないです。クライアントは、リソースサーバにアクセストークンを渡すことにより、保護リソースにアクセスできるようになる [13]。OAuth2.0 認証を図 9 に示す。

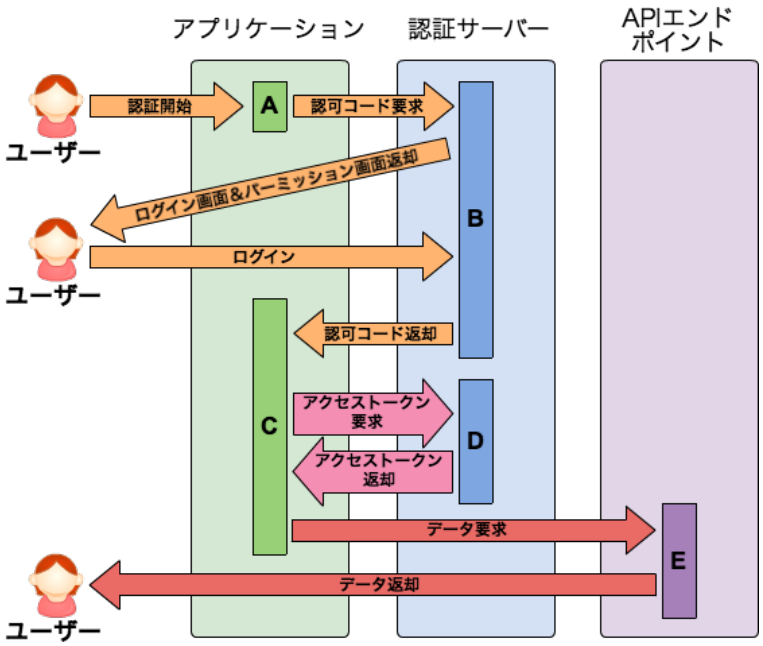

図 6　OAuth2.0 認証の流れ図。infoScoop 開発者ブログ・Java で OpenID Connect の Basic Client を実装してみた [14] より引用

3.3.2 方法

Charge HR を正しく手首につけることで自動的に脈拍・歩数・移動距離・消費エネルギー・階段数のデータを記録することができる。携帯端末と同期することで記録されたデータを見ることがでる。本システムは記録されたデータの中に必要とする脈拍データの FitbitAPI [15] を 30 分毎に分単位で Fitbit.com から取得し、脈拍数の観察と分析をする。不整脈には表 1 のように徐脈 (脈が異常に遅くなる)、頻脈 (脈が異常に早くなる)、期外収縮

(時々脈が飛ぶ) の 3 つに分類できる。この 3 種類の脈拍特徴をデータ化にし、取得した脈拍データと照らし合わせることで一致するかもしくは相似するパラメータが検出された場合には携帯電話にインストールされた Gmail アプリに通知する。

3.3.3 動作

Fitbit アプリがインストールされた携帯電話端末の Bluetooth とアプリ内の [常に同期] をオンにすると Charge HR は 30 分毎に自動同期をする。システムの流れを図 10 に示す。

自動同期をオフにする場合は動作 1 (手動でデータを同期する)
自動同期をオンにする場合は動作 2 (30 分毎に自動でデータを同期する)

動作 1

1. Charge HR と Fitbit アプリを同期できる状態にする。
2. Charge HR を身につけた状態且つ機器が正常に動作し、Bluetooth をオンにする。
3. cron を使い、指定時刻（最短時刻は 2 分）毎に main.py を起動する。
4. cron で指定した時間より以前に fitbit アプリを開き、手動同期をする。
5. 不整脈が判断された場合にはメールが送信される。

動作 2

1. Charge HR と Fitbit アプリを同期できる状態にする。（同上）
2. Charge HR を身につけた状態且つ機器が正常に動作できるのを確認する。（同上）
3. 携帯電話の Bluetooth とアプリ内の [常に同期] をオンにする。
4. cron を使い、自動同期する時刻 + 1 分（31 分）毎に main.py を起動する。
5. 不整脈が判断された場合にはメールが送信される。（同上）

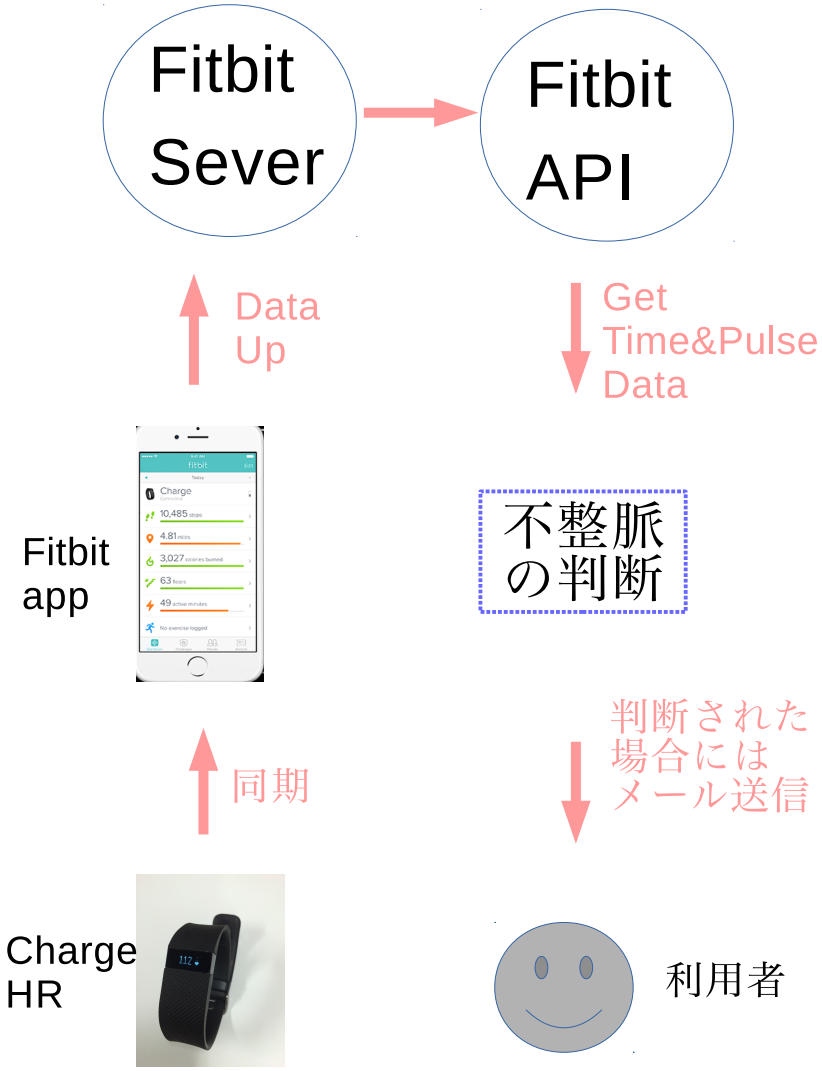

図 7　システムの流れ。30 分毎に Charge HR が Fitbit app を通じて Fitbit server にデータが送信される。Fitbit API を使い、時間と脈拍数のデータだけを取得する。取得したデータを元に不整脈か否かを判定し、不整脈であれば利用者に通知メールが送信される。

4　ハードウェアの詳細

本章では、本研究で用いたウェアラブル機器 Charge HR について解説をする。

4.1　Charge HR

Charge HR は、エクササイズやその他の運動中に、心拍数とアクティビティを自動的に記録する高度なリストバンド型心拍記録計である。Charge HR 本体を図 1、図 2 に示す。

図 8　Charge HR の表面。ディスプレイにはリアルタイムの脈拍数が表示される

図 9　Charge HR の背面。内側に見える 2 つの点が脈拍を測るための LED ライトである

4.1.1　心拍数の計測

心臓が動くと、血液量の変化により、毛細血管が拡張・収縮する。Charge HR の背面にある PurePulse^{TM} (ピュアパルス)[*2] LED ライトは肌に反射することで、血液量の変化を検出し、細かく調整されたアルゴリズムが適用され、心拍数を継続的に自動測定する。デフォルトでは、Charge HR を手首に着用した時に心拍数センサーがアクティブになる。

4.1.2　心拍数ゾーン

心拍数ゾーンは 4 種類がある。異なるトレーニングの強度を目標にすることで、最適なワークアウトをすることに役立つ。心拍数ゾーンは、カルボーネン公式による最大心拍数（220 - 年齢）をもとに計算される。Charge HR のハートアイコンを図 3 に示す。これを見ると心拍数ゾーンをすばやく確認できる [4]。

・ゾーン圏外（最大心拍数の 50% 以下）は、心拍数は上昇していても運動のレベルに達していない状態を表す。

[*2] PurePulse とは PurePulse は、健康データやトレーニングの激しさを 1 日中モニタリングする、唯一の自動的、継続的なリストバンド型テクノロジーです。 連続計測、長時間動作を可能にした心拍センサーにより、一日中心拍数を計測できるよう特別に設計されている。

図 10　心拍数ゾーンの種類。左側からゾーン圏外、脂肪消費、有酸素運動、ピークゾーンである。Charge HR 製品マニュアル [4] より引用

・ 脂肪燃焼ゾーンは、心拍数が最大 50~60% の間にあることを意味し、低 ~ 中の強度のエクササイズです。高い割合でカロリーが脂肪から燃焼されることから、脂肪燃焼と呼ばれているが全体のカロリー消費は低いです。

・ 有酸素運動ゾーンは、心拍数が最大 70~84% の間にあることを意味し、中 ~ 高の強度のエクササイズゾーンです。このゾーンは全力まではいかないものの、かなり頑張る必要がある。

・ ピークゾーンは、心拍数が最大 85% 以上あることを意味し、非常に強度なエクササイズゾーンです。

5　結果と考察

システムの検証は複数の機器で同様な結果が得られるかを確認するため 2 台の Charge HR を使用した。不整脈の一種である頻脈（脈拍数が 150 回/min 以上）を実際に運動をすることで脈拍数が上昇し、頻脈を再現をした。又、脈拍データファイルに頻脈と徐脈（脈拍数が 40 回/min 以下）のデータを用意し、実際に動作が正常に作動するかを確認した。表 3 に示す。2 種類 (テストデータと API データ) の頻脈出力メールの通知を確認すると本システムは正常に異常通知をすることができた。図 14 と図 11 に示す。それ以外に急に頻脈のメール通知を受信した場合に対応ができないことを想定し、頻脈になる直前に脈拍数が 120 回 ~149 回/min の間になると事前に注意通知も届いた。図 15 に示す。これらの結果を見ると本システムは正常に徐脈、頻脈、頻脈になる直前を検知し、異常通知をすることができた。

5.1　検証

5.1.1　出力結果

頻脈は 1 機器 (筆者が普段から身につける機器) と 2 機器（テスト機器）の運動による頻脈を到達し、受信したメール通知を図 11, 図 12 に示す。

図 11　1 機器による頻脈の受信メール。頻脈である脈拍数 151 回/min が確認できる

図 12　2 機器による頻脈の受信メール。頻脈である脈拍数 152 回/min が確認できる

徐脈は通常徐脈になることが困難なため実際にテストデータを用意した。表 3 のように示す。

脈拍数
39 32 13 48 177

表 3　用意したテストデータ。脈拍数が 39 回,32 回,13 回/min が徐脈。脈拍数 177 回/min が頻脈

表 3 のテストデータを用いてシステムを作動すると受信したメールに脈拍数が 39 回,32 回,13 回/min の徐脈だけ正確に出力された。図 13 に示す。

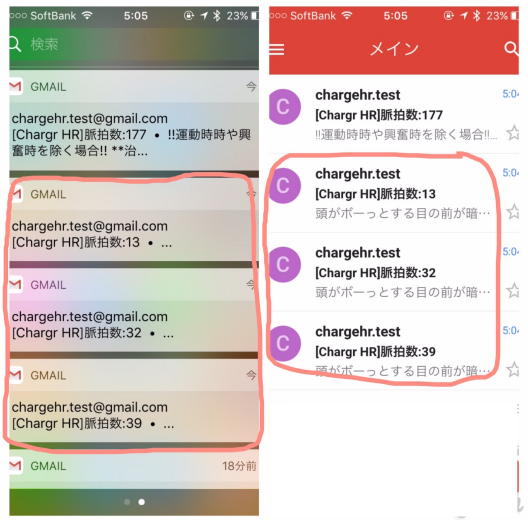

図 13　テストデータを使用した徐脈の受信メール。脈拍数が 39 回,32 回,13 回/min の徐脈が確認できる

又、表 3 のテストデータを用いて受信したメールに脈拍数が 177 回/min の頻脈が正確に出力された。図 14 に示す。

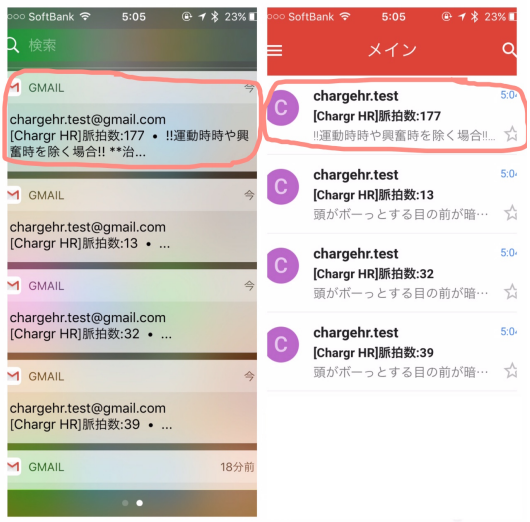

図 14　テストデータを使用した頻脈の受信メール。脈拍数が 177/回 min の頻脈が確認できる

頻脈になる直前に脈拍数が 120 回 ~149 回/min の間に受信されたメール内容。図 15 に示す。

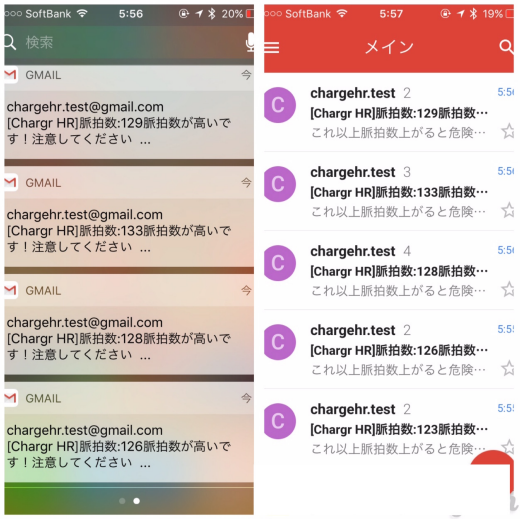

図 15　頻脈より直前に脈拍数 120 回/min を超えた時の受信メール。上から脈拍数 129 回,133 回,128 回,126 回/min が確認できる

5.1.2　脈拍データの分析

Charge HR をつけた状態で脈拍データが記録され、1 日毎の脈拍データが取得できるようになる。取得したデータを 1 日、1 週間、1 ヶ月、3 ヶ月、6 ヶ月、曜日毎に分け、1 日の中でも活動する AM9:00 から PM18:00 までの時間帯だけを時系列グラフにした。図 16、図 17、図 25、図 26、図 27、図 18～ 図 24 に示す。
時系列グラフの x 軸は時間、y 軸は心拍数 (回/min)

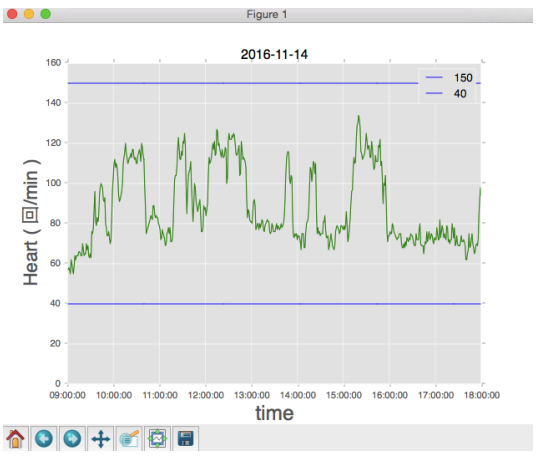

図 16　1 日の時系列グラフ

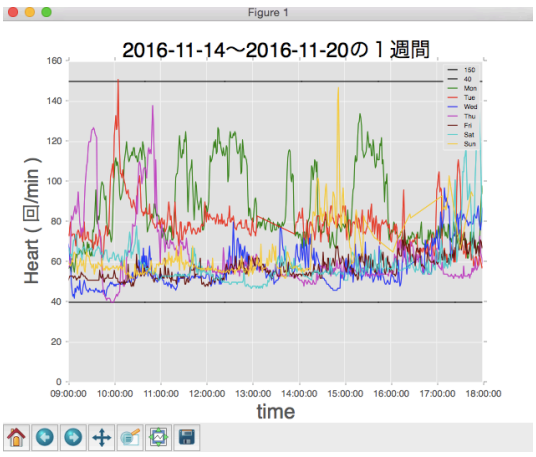

図 17　1 週間のデータを曜日毎に色分けた時系列グラフ

図 16, 図 17 のように 1 日毎、1 週間毎に分けることでどの時間帯に脈拍数が上がるかがわかる。

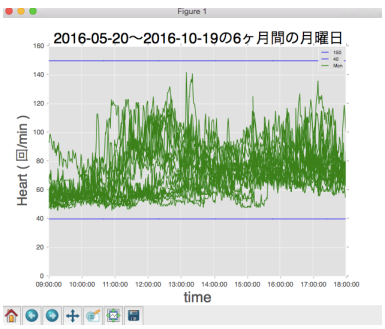

図 18　6 ヶ月間の月曜日だけの時系列グラフ

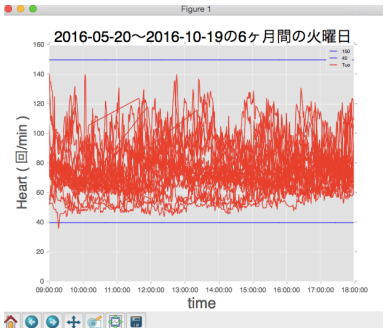

図 19　6 ヶ月間の火曜日だけの時系列グラフ

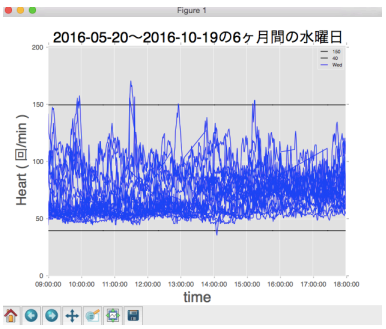

図 20　6 ヶ月間の水曜日だけの時系列グラフ

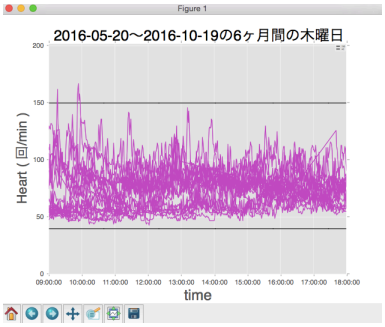

図 21　6 ヶ月間の木曜日だけの時系列グラフ

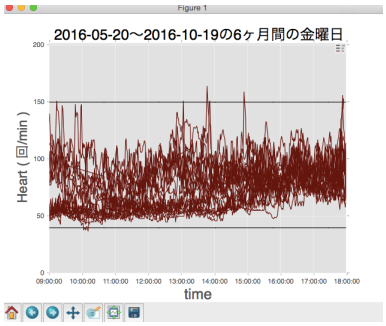

図 22　6 ヶ月間の金曜日だけの時系列グラフ

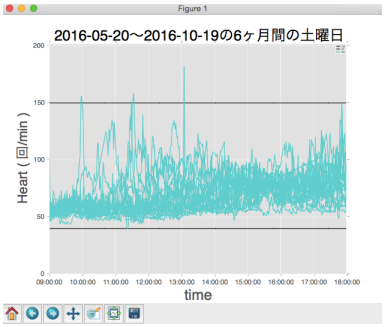

図 23　6 ヶ月間の土曜日だけの時系列グラフ

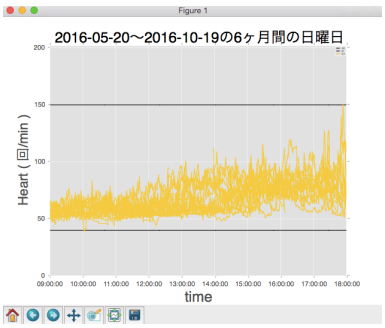

図 24　6 ヶ月間の日曜日だけの時系列グラフ

図 18～ 図 24 のように曜日毎に多数のデータの線が重ね、太くなっている部分（太線）があり、図 18 の 11:00 と図 22 の 9:00 から 1 本の太線から 2 つに分かれているのをグラフから読み取ることができる。このように 2 つに分かれると 2 種類の行動パターンがある。実際に機器をつけた筆者は図 18 の月曜日と図 22 の金曜日に学校に行くこと (上昇の線) と家にいる（平の線）ことが多いため実際の行動がグラフに現れた。又、図 23 の土曜日や図 24 の日曜日のように大きく 2 つに分かれることがすくない土曜日や日曜日は家に入る事が多い。このように分かれることで行動パターンがわかる。

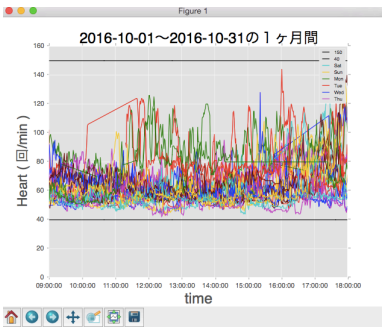

図 25　1 ヶ月間の時系列グラフ

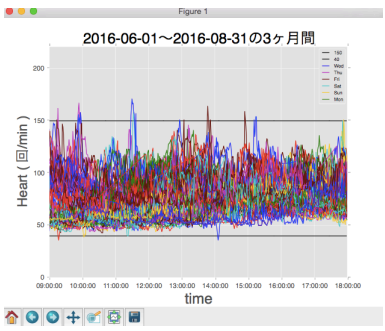

図 26　3 ヶ月間の時系列グラフ

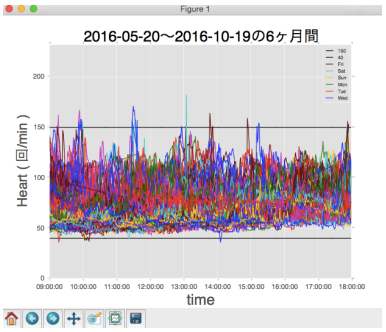

図 27　6 ヶ月間の時系列グラフ

図 25～ 図 27 のように月曜日から日曜日までのデーてを 1 ヶ月間、3 ヶ月間、6 ヶ月間に分かれるとデータの量が多く、徐脈や頻脈の数だけ確認する事ができる。

5.1.3　考察

1 週間毎の時系列グラフを見るとどの曜日に脈拍数が高いかがわかり、曜日毎に分かれるとその曜日の行動パターンとグラフ内の太線が上昇する場合に脈拍数の上昇傾向があるとはわかる。月毎で見ると毎月のデータが増える毎に徐脈や頻脈の増加頻度がわかる。このように曜日毎、月毎に分かれると異なる情報が得られ、脈拍数が高くなる時を予想することができる。もし徐脈、頻脈が出た時は事前に準備や対応もしやすくなる。

5.2　問題点

問題点は大きく 2 点挙げられる。1 点目は不整脈は中年以上の人に起こりやすいので対象者がお年寄りの方が多くなる。本システムでは使用可能までの環境構築はお年寄りの方に不向である。又、使いやすさに欠けることです。本システムは cron を用いたので PC 環境に依存した設定などが不便である点。2 点目はリアルタイムに脈拍数 API が得られにくいので万が一に不整脈が起きた場合に逸早くメールでの通知ができない点。

5.3　今後の課題

上記の問題点についての今後の課題を述べる

1 点目の環境構築と使いやすさの改善方法はローカル環境を web で作動する環境に変えること。そうすると誰でも使用可能より簡単な設定ができ、PC 環境にも依存しなくなる。web 環境の構築が改善に不可欠であると考える。2 点目の逸早く脈拍数の API 取得不可の改善方法は Ftibit アプリ開発側に同期頻度の改善要望をし、改善して頂けるようにすべきであると考える。

6　結論

本研究では、身近に不整脈患者がいないため運動で不整脈の一種である頻脈を再現した。システム検証した結果、運動で脈拍数を 150 回/min 以上出した場合に通知メールが利用者に届いた。又、徐脈は再現することが困難なため用意した徐脈と頻脈データを用いて検証テストをした。結果は、2 種類の不整脈ともにメールの通知が届いた。しかし、Fitbit アプリ側が 30 分毎にデータの同期を行うため突然不整脈が発症した場合に即急に自動で利用者にメールで通知することができなかった。手動でデータ同期した場合にはすぐにメールの通知が届いた。

本システムは脈拍で判る持病の早期発見・予防については具体的な持病を見える化にすることができなかった。しかし、脈拍データを用いて時系列グラフにした結果、脈拍数の変動は時系列グラフから読み取ることが可能である。又、長期的に多くのデータが取得可能であるため分析手法や持病の脈拍の医療データを取得可能であれば脈拍パラメータをデータと照らし合わせることで可能と言える。

本システムは簡易に不整脈を発見するため医療機器に比べると精度等が圧倒的に劣るため、不整脈を検知した場合には医療機関での検査・治療が必要になる。費用等は表 2 を参考すると健康保険の適応があるため高額医療申請の手続きをすると約 10 万円の自己負担となる。

謝辞

本研究を進めるにあたり、様々なご指導ご意見を頂いた大垣斉准教授、情報教育システム研究室のメンバーの方々に深く感謝いたします。また、システムの検証にご協力して頂いた中谷元君に感謝致します。

参考文献

[1] 2016 年版高齢社会白書. http://www8.cao.go.jp/kourei/whitepaper/w-2016/gaiyou/pdf/1s1s.pdf

[2] 平成 26 年度厚生労働省医療費の動向調査. http://www.mhlw.go.jp/topics/medias/year/14/dl/iryouhi_data.pdf

[3] 公益社団法人日本脳卒中協会心房細動週間・脈の日. http://www.jsa-web.org/pulse/index.html

[4] Charge HR 製品マニュアルバージョン 1.0. http://staticcs.fitbit.com/content/assets/help/manuals/manual_charge_hr_ja.pdf

[5] 国立循環器病研究センター循環器病情報サービス. http://www.ncvc.go.jp/cvdinfo/pamphlet/heart/pamph06.html

[6] Boston Scientific. http://www.bostonscientific.com/jp-JP/health-conditions/ihb/ihb-05.html

[7] 看護 roo. https://www.kango-roo.com/sn/k/view/1501

[8] QLife・負荷心電図検査. https://www.qlife.jp/dictionary/exam/item/i_01100/

[9] QLife・心臓超音波検査. https://www.qlife.jp/dictionary/exam/item/i_01300/

[10] KOMPS・心臓電気生理学的検査. http://kompas.hosp.keio.ac.jp/contents/000434.html

[11] 北岡クリニック・不整脈. http://www.kitaoka-clinic.com/pulse/

[12] 筑波メディカルセンター病院・入院費用概算. http://www.tmch.or.jp/hosp/examination/hospitalization/expenses.html

[13] murashun.jp・OAuth2.0 の仕組みと認証方法. https://murashun.jp/blog/20150920-01.html

[14] infoScoop 開発者ブログ・Java で Open ID Connect の Basic Client を実装してみた. https://www.infoscoop.org/blogjp/tag/oauth2-0/

[15] FitbitAPI と Python で心拍数時系列データの取得方法. http://blog.mr-but-dr.xyz/programming/fitbit-python-heartrate-howto/

付録 A　ソースコード

A.1　main.py

```
#!/Usr/bin/env python
# -*- coding: utf-8 -*-

""" Fitbit """
import fitbit
import gather_keys_oauth2 as Oauth2
import matplotlib.pyplot as plt
import numpy as np

""" Gmail """
import os.path
import datetime
import smtplib
from email import Encoders
from email.Utils import formatdate
from email.MIMEBase import MIMEBase
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText



""" for OAuth2.0 """
USER_ID = ''          #Charge HR の USER_ID 入力
CLIENT_SECRET = ''    # CLIENT_SECRET 入力


""" for obtaining Access-token and Refresh-token """
server = Oauth2.OAuth2Server(USER_ID, CLIENT_SECRET)
server.browser_authorize()
print('FULL RESULTS = %s' % server.oauth.token)
print('ACCESS_TOKEN = %s' % server.oauth.token['access_token'])

ACCESS_TOKEN = server.oauth.token['access_token']
REFRESH_TOKEN = server.oauth.token['refresh_token']


""" Authorization """
auth2_client = fitbit.Fitbit(USER_ID, CLIENT_SECRET, oauth2=True,
```

```
                access_token=ACCESS_TOKEN, refresh_token=REFRESH_TOKEN)

""" Getting data (自動時間指定)"""

Today = datetime.datetime.today()
date_now = Today.strftime('%H:%M')
date_format = '%H:%M'
d = datetime.datetime.strptime(date_now, date_format)
d -= datetime.timedelta(minutes = 30)  #minutes>1
date_before = d.strftime(date_format)

fitbit_stats = auth2_client.intraday_time_series('activities/heart',
base_date=Today, detail_level='1min',start_time= date_before ,
end_time= Today.strftime('%H:%M') )

print "----------------------------------------"
print Today.strftime('%Y-%m-%d %H:%M:%S')
print "取得開始時刻    取得終了時刻"
print date_before,
print "\t\t%s"%Today.strftime('%H:%M')



""" Getting data (手動時間指定)"""
"""
fitbit_stats = auth2_client.intraday_time_series('activities/heart',
base_date='2016-11-15', detail_level='1min',start_time="09:00",end_time='18:00')
"""
#取得したい日付、開始時刻、終了時刻に変更



""" Getting only 'heartrate' and 'time' """
stats = fitbit_stats['activities-heart-intraday']['dataset']

""" Timeseries data of Heartrate """
f1 = open('dataHR-timeseries.txt', 'w')
HR = []
for var in range(0,len(stats)):
    f1.write(str(stats[var]['value']) )
    f1.write("\t")
    f1.write(stats[var]['time'])
    f1.write("\n")
```

```
    HR = HR + [stats[var]['value']]
f1.close()

#write data
f2 = open('dataHR-heart.txt', 'w')
H = []
for var in range(0,len(stats)):
    f2.write(str(stats[var]['value']) )
    f2.write("\n")
    H = H + [stats[var]['value']]
f2.close()

#write time
f3 = open('dataHR-time.txt', 'w')
R = []
for var in range(0,len(stats)):
    f3.write(str(stats[var]['time']) )
    f3.write("\n")
    R = R + [stats[var]['value']]
f3.close()



f4 = open('dataHR-timeseries.txt','r')
for data_heart_time in open('dataHR-timeseries.txt','r'):
    #time_heard = data_heart_time.split('\t')
    f4.close()
    ##不整脈##
    if len(data_heart_time)<=12 and data_heart_time[0]<='3'and
        data_heart_time[1]<='9':
            print "脈拍数:",
            print data_heart_time[0],data_heart_time[1],
            print ' 徐脈'
            print "発生時間: ",
            print data_heart_time[3],data_heart_time[4],data_heart_time[5],\
                  data_heart_time[6],data_heart_time[7],data_heart_time[8],\
                  data_heart_time[9],data_heart_time[10]

    if len(data_heart_time)<=13 and data_heart_time[0]>='1' and
        data_heart_time[1]<='4':
        if data_heart_time[0]=='1' and data_heart_time[1]>='2':
            print "脈拍数:",
            print data_heart_time[0],data_heart_time[1],data_heart_time[2],
```

```
            print "発生時間: ",
            print data_heart_time[4],data_heart_time[5],data_heart_time[6],\
                  data_heart_time[7],data_heart_time[8],data_heart_time[9],\
                  data_heart_time[10],data_heart_time[11]


    elif data_heart_time[0]>='1' and data_heart_time[1]>='5' and
          data_heart_time[2]>='0':
            print "脈拍数:",
            print data_heart_time[0],data_heart_time[1],data_heart_time[2],
            print ' 頻脈'
            print "発生時間: ",
            print data_heart_time[4],data_heart_time[5],data_heart_time[6],\
                  data_heart_time[7],data_heart_time[8],data_heart_time[9],\
                  data_heart_time[10],data_heart_time[11]


""" Send to the Gmail """
#Gmail アカウント
ADDRESS = "   @gmail.com"      #GmailID 入力
PASSWARD = "  "               # PW 入力

#SMTP サーバの設定 (Gmail 用)
SMTP = "smtp.gmail.com"
PORT = 587


def create_message(from_addr, to_addr, subject, body, mime=None,
    attach_file=None):
    """
    メッセージを作成する
    @:param from_addr 差出人
    @:param to_addr 宛先
    @:param subject 件名
    @:param body 本文
    @:param mime MIME
    @:param attach_file 添付ファイル
    @:return メッセージ
    """
    msg = MIMEMultipart()
    msg["From"] = from_addr
    msg["To"] = to_addr
    msg["Date"] = formatdate()
```

```
    msg["Subject"] = subject
    body = MIMEText(body)
    msg.attach(body)

    # 添付ファイル
    if mime != None and attach_file != None:
        attachment = MIMEBase(mime['type'],mime['subtype'])
        file = open(attach_file['path'])
        attachment.set_payload(file.read())
        file.close()
        Encoders.encode_base64(attachment)
        msg.attach(attachment)
        attachment.add_header("Content-Disposition","attachment",
        filename=attach_file['name'])

    return msg

def send(from_addr, to_addrs, msg):
    """
    メールを送信する
    @:param from_addr 差出人
    @:param to_addr 宛先 (list)    @:param msg メッセージ
    """
    smtpobj = smtplib.SMTP(SMTP, PORT)
    smtpobj.ehlo()
    smtpobj.starttls()
    smtpobj.ehlo()
    smtpobj.login(ADDRESS, PASSWARD)
    smtpobj.sendmail(from_addr, to_addrs, msg.as_string())
    smtpobj.close()



if __name__ == '__main__':
    #宛先アドレス入力
    to_addr = "    "

    #件名と本文
    #subject
    #body

f4 = open('dataHR-timeseries.txt','r')
for data_heart_time in open('dataHR-timeseries.txt','r'):
    f4.close()
```

```
if (os.path.getsize("dataHR-timeseries.txt"))==0:
    print "API 取得不能、アプリを同期してください"
    exit()

if len(data_heart_time)==12 and data_heart_time[0]<='3'and
    data_heart_time[1]<='9':
    a=data_heart_time[0],data_heart_time[1]
    subject = "[Chargr HR] 脈拍数:%s%s"%a
    body = "頭がボーっとする\n 目の前が暗くなる\n 手足が冷たい\
            \n むくむ\n 疲れやすい\n 失神する\n\t などの症状はないか"
    msg = create_message(ADDRESS, to_addr, subject, body)
    send(ADDRESS, [to_addr], msg)

if len(data_heart_time)<=13 and data_heart_time[0]>='1' and
    data_heart_time[1]<='4':
    if len(data_heart_time)<=13 and data_heart_time[0]=='1' and
        data_heart_time[1]>='2':
       b = data_heart_time[0],data_heart_time[1],data_heart_time[2]
       subject = "[Chargr HR] 脈拍数:%s%s%s 脈拍数が高いです！注意してください"%b
       body = "これ以上脈拍数上がると危険です"
       msg = create_message(ADDRESS, to_addr, subject, body)
       send(ADDRESS, [to_addr], msg)


if len(data_heart_time)<=13 and data_heart_time[0]>='1' and
    data_heart_time[1]>='5' and data_heart_time[2]>='0':
    """
    #添付ファイル設定（text.txt ファイルを添付）
    #mime={'type':'text', 'subtype':'comma-separated-values'}
    #attach_file={'name':'test.txt', 'path':'./text.txt'}

    #メッセージの作成（添付ファイルあり）
    #msg = create_message(ADDRESS, to_addr, subject, body, mime, attach_file)
    #メッセージ作成（添付ファイルなし）
    #msg = create_message(ADDRESS, to_addr, subject, body)
    #送信
    #send(ADDRESS, [to_addr], msg)
    """
    c = data_heart_time[0],data_heart_time[1],data_heart_time[2]
    subject = "[Chargr HR] 脈拍数:%s%s%s"%c
    body = "!!運動時時や興奮時を除く場合!!\n**治療が必要です**\
            \n\n 脈が速くなり、どきどきが続く\
```

```
                \n 頻脈の結果血圧が低下の場合は失神する\
                \n 頻脈時短時間胸が痛くなる\n\t などの症状\
                \n\n 安静にしているときに頻脈が急に起きると不安になるものだが、\
                \n 脈拍数が 120 以下で、安静にしていると徐々に普通の脈拍に戻る\
                \n ような場合はほとんどの場合病的な頻脈ではないので、\
                \n 脈拍が早くなってしばらくの間続いても、“120 以下で規則正しく脈が\
                \n 打っているから大丈夫”と自分自身で納得して落ち着く事が大切である。"
        msg = create_message(ADDRESS, to_addr, subject, body)
        send(ADDRESS, [to_addr], msg)

"""確認"""
HRmax = np.max(HR)
HRmin = np.min(HR)
print ("脈拍回数:%s\n")%len(stats),("最大脈数:%s\n")%HRmax,
        ("最小脈数:%s\n")%HRmin,("脈拍:%s\n")%HR
print "----------------------------------------"

"""棒グラフ表示
plt.hist(HR, bins=len(stats), range=(HRmin,HRmax))
plt.xlabel('Heart')
plt.ylabel('Number of times')
plt.show()
"""


"""時系列グラフ表示"""
plt.style.use('ggplot')
font = {'family' : 'meiryo'}
plt.rc('font', **font)

i=[]
j=[]
l=[]
m=[]
f5 = open('dataHR-time.txt', 'r')
for data in open('dataHR-time.txt','r'):
    heart = data.split('\n')
    f5.close()
    h = heart[0].split('\n')
    i=i+h

f6 = open('dataHR-heart.txt', 'r')
for data in open('dataHR-heart.txt','r'):
```

```
    time = data.split('\n')
    f6.close()
    t = time[0].split('\n')
    j=j+t

for k in range(len(i)):
    k=[40]
    l=l+k
for k in range(len(i)):
    k=[150]
    m=m+k

prop = fm.FontProperties(fname='/usr/share/fonts/ja/TrueType/
                          kochi-gothic-subst.ttf')
plt.plot(pd.to_datetime(i),m,"b",label = "150")
plt.plot(pd.to_datetime(i),l,"b",label = "40")
plt.legend()
if HRmax<150:
    h=160
    plt.ylim(0,h)
else:
    plt.ylim(0,HRmax+50)
plt.plot(pd.to_datetime(i),j)
plt.title("%s"%Today.strftime('%Y-%m-%d'))
plt.xlabel('time', size='22')
plt.ylabel(u'Heart ( 回/min )', size='20',fontproperties=prop)
plt.show()
```

A.2 analysis_day.py

```
#!/Usr/bin/env python
# -*- coding: utf-8 -*-

""" Fitbit """
import fitbit
import gather_keys_oauth2 as Oauth2
import matplotlib.pyplot as plt
import numpy as np

""" Gmail """
import os.path
import datetime
import smtplib
```

```
from email import Encoders
from email.Utils import formatdate
from email.MIMEBase import MIMEBase
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText

import time
import pandas as pd
import matplotlib.font_manager as fm
import os



""" for OAuth2.0 """
USER_ID = ''          #Charge HR の USER_ID 入力
CLIENT_SECRET = ''    # CLIENT_SECRET 入力



""" for obtaining Access-token and Refresh-token """
server = Oauth2.OAuth2Server(USER_ID, CLIENT_SECRET)
server.browser_authorize()

print('FULL RESULTS = %s' % server.oauth.token)
print('ACCESS_TOKEN = %s' % server.oauth.token['access_token'])

ACCESS_TOKEN = server.oauth.token['access_token']
REFRESH_TOKEN = server.oauth.token['refresh_token']



""" Authorization """
auth2_client = fitbit.Fitbit(USER_ID, CLIENT_SECRET, oauth2=True,
access_token=ACCESS_TOKEN, refresh_token=REFRESH_TOKEN)

day = datetime.datetime.strptime('2016-11-24','%Y-%m-%d')
#取得する日付に変更



""" Getting data """

fitbit_stats = auth2_client.intraday_time_series('activities/heart', day,
                      detail_level='1min',start_time="09:00",end_time='18:00')
#開始時刻と終了時刻に変更
```

```
""" Getting only 'heartrate' and 'time' """
stats = fitbit_stats['activities-heart-intraday']['dataset']


""" Timeseries data of Heartrate """

txt_name = "{:%Y%m%d}.txt".format(day)

f1 = open(txt_name, 'w')
HR = []
for var in range(0,len(stats)):
    f1.write(str(stats[var]['value']) )
    f1.write("\t")
    f1.write(stats[var]['time'])
    f1.write("\n")
    HR = HR + [stats[var]['value']]
f1.close()
#脈拍データだけ書き込み
f2 = open("{:%Y%m%d}-heart.txt".format(day), 'w')
H = []
for var in range(0,len(stats)):
    f2.write(str(stats[var]['value']) )
    f2.write("\n")
    H = H + [stats[var]['value']]
f2.close()

#時間データだけ書き込み
f3 = open("{:%Y%m%d}-time.txt".format(day), 'w')
R = []
for var in range(0,len(stats)):
    f3.write(str(stats[var]['time']) )
    f3.write("\n")
    R = R + [stats[var]['value']]
f3.close()


"""確認"""
HRmax = np.max(HR)
HRmin = np.min(HR)
print day
print "----------------------------------------"
print ("脈拍回数:%s\n")%len(stats),("最大脈数:%s\n")%HRmax,("最小脈数:%s\n")%HRmin
```

```
print "----------------------------------------"

"""棒グラフ表示
plt.hist(HR, bins=len(stats), range=(HRmin,HRmax))
plt.xlabel('Heart')
plt.ylabel('Number of times')
plt.show()
"""


"""時系列グラフ表示"""
plt.style.use('ggplot')
font = {'family' : 'meiryo'}
plt.rc('font', **font)

i=[]
j=[]
l=[]
m=[]
f4 = open("{:%Y%m%d}-time.txt".format(day), 'r')
for data in open("{:%Y%m%d}-time.txt".format(day), 'r'):
    heart = data.split('\n')
    f4.close()
    h = heart[0].split('\n')
    i=i+h

f5 = open("{:%Y%m%d}-heart.txt".format(day), 'r')
for data in open("{:%Y%m%d}-heart.txt".format(day), 'r'):
    time = data.split('\n')
    f5.close()
    t = time[0].split('\n')
    j=j+t

for k in range(len(i)):
    k=[40]
    l=l+k
for k in range(len(i)):
    k=[150]
    m=m+k
prop = fm.FontProperties(fname='/usr/share/fonts/ja/TrueType/
        kochi-gothic-subst.ttf')
plt.plot(pd.to_datetime(i),m,"k",label = "150")
plt.plot(pd.to_datetime(i),l,"k",label = "40")
```

```
plt.legend()
if HRmax<150:
    h=160
    plt.ylim(0,h)
else:
    plt.ylim(0,HRmax+50)
plt.xlim(pd.to_datetime(i)[0],pd.to_datetime(i)[-1])
plt.plot(pd.to_datetime(i),j,"green")
plt.title("%s"%day.strftime('%Y-%m-%d'))
plt.xlabel('time', size='22')
plt.ylabel(u'Heart ( 回/min )', size='20',fontproperties=prop)
plt.show()
```

A.3 analysis_week.py

```
#!/Usr/bin/env python
# -*- coding: utf-8 -*-

""" Fitbit """
import fitbit
import gather_keys_oauth2 as Oauth2
import matplotlib.pyplot as plt
import numpy as np

""" Gmail """
import os.path
import datetime
import smtplib
from email import Encoders
from email.Utils import formatdate
from email.MIMEBase import MIMEBase
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText

import time
import pandas as pd
import matplotlib.font_manager as fm
import os

""" for OAuth2.0 """
USER_ID = ''          #Charge HR の USER_ID 入力
```

```
CLIENT_SECRET = ''     # CLIENT_SECRET 入力


""" for obtaining Access-token and Refresh-token """
server = Oauth2.OAuth2Server(USER_ID, CLIENT_SECRET)
server.browser_authorize()

print('FULL RESULTS = %s' % server.oauth.token)
print('ACCESS_TOKEN = %s' % server.oauth.token['access_token'])

ACCESS_TOKEN = server.oauth.token['access_token']
REFRESH_TOKEN = server.oauth.token['refresh_token']



""" Authorization """
auth2_client = fitbit.Fitbit(USER_ID, CLIENT_SECRET, oauth2=True,
access_token=ACCESS_TOKEN, refresh_token=REFRESH_TOKEN)

day = datetime.datetime.strptime('2016-11-24','%Y-%m-%d')
#取得する日付に変更


""" Getting data """

fitbit_stats = auth2_client.intraday_time_series('activities/heart', day,
                        detail_level='1min',start_time="09:00",end_time='18:00')
#開始時刻と終了時刻に変更

""" Getting only 'heartrate' and 'time' """
stats = fitbit_stats['activities-heart-intraday']['dataset']


""" Getting only 'heartrate' and 'time' """
""" Getting only 'heartrate' and 'time' """
stats = fitbit_stats['activities-heart-intraday']['dataset']



txt_name = "{:%Y%m%d}.txt".format(day)

for i in range(0,7):
    locals()["D%d" % i ]=day + datetime.timedelta(days=i)
```

```
f999 = open(txt_name, 'w')
HR = []
for var in range(0,len(stats)):
    f999.write(str(stats[var]['value']) )
    f999.write("\t")
    f999.write(stats[var]['time'])
    f999.write("\n")
    HR = HR + [stats[var]['value']]
f999.close()


"""時系列グラフ表示"""
plt.style.use('ggplot')
font = {'family' : 'meiryo'}
plt.rc('font', **font)

i0=[]
j0=[]
l=[]
m=[]
for ii in range(1,7):
    locals()["i%d" % ii ]=[]
    locals()["f%d" % ii ] = open("{:%Y%m%d}-time.txt".format
                                 (locals()["D%d" % ii ]), 'r')
    for data in open("{:%Y%m%d}-time.txt".format(locals()["D%d" % ii ]), 'r'):
        heart = data.split('\n')
        locals()["f%d" % ii ].close()
        locals()["h%d" % ii ] = heart[0].split('\n')
        locals()["i%d" % ii ]=locals()["i%d" % ii ]+locals()["h%d" % ii ]

f1 = open("{:%Y%m%d}-time.txt".format(day), 'r')
for data in open("{:%Y%m%d}-time.txt".format(day), 'r'):
    heart = data.split('\n')
    f1.close()
    h = heart[0].split('\n')
    i0=i0+h

for i in range(1,7):
    locals()["j%d" % i ]=[]
    locals()["ff%d" % i ] = open("{:%Y%m%d}-heart.txt".format
```

```
                                (locals()["D%d" % i ]), 'r')
    for data in open("{:%Y%m%d}-heart.txt".format(locals()["D%d" % i ]), 'r'):
        time = data.split('\n')
        locals()["ff%d" % i ].close()
        locals()["t%d" % i ] = time[0].split('\n')
        locals()["j%d" % i ]=locals()["j%d" % i ]+locals()["t%d" % i ]

ff1 = open("{:%Y%m%d}-heart.txt".format(day), 'r')
for data in open("{:%Y%m%d}-heart.txt".format(day), 'r'):
    time = data.split('\n')
    ff1.close()
    t = time[0].split('\n')
    j0=j0+t


print "----------------------------------------"
print day.strftime('%Y-%m-%d'),"?%s"%D6.strftime('%Y-%m-%d')
print "----------------------------------------"


for k in range(len(locals()["i%d" % ii ])):
    k=[40]
    l=l+k
for k in range(len(locals()["i%d" % ii ])):
    k=[150]
    m=m+k

prop = fm.FontProperties(fname='/usr/share/fonts/ja/TrueType/
                                kochi-gothic-subst.ttf')
plt.plot(pd.to_datetime(locals()["i%d" % ii ]),m,"k",label = "150")
plt.plot(pd.to_datetime(locals()["i%d" % ii ]),l,"k",label = "40")
for i in range(0,7):
    if locals()["D%d" % i ].isoweekday()==1:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "green",label = "Mon")
    if locals()["D%d" % i ].isoweekday()==2:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "orangered",label = "Tue")
    if locals()["D%d" % i ].isoweekday()==3:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "b",label = "Wed")
    if locals()["D%d" % i ].isoweekday()==4:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
```

```
        "m",label = "Thu")
    if locals()["D%d" % i ].isoweekday()==5:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "#660000",label = "Fri")
    if locals()["D%d" % i ].isoweekday()==6:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "#00CCCC",label = "Sat")
    if locals()["D%d" % i ].isoweekday()==7:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "#FFCC00",label = "Sun")
plt.legend(fontsize=7)

Ymax=[]
for i in range(0,7):
    locals()["y%d" % i ]=map(int,locals()["j%d" % i ])
    Ymax=Ymax+locals()["y%d" % i ]
ymax=max(Ymax)

if ymax<150:
    h=160
    plt.ylim(0,h)
else:
    plt.ylim(0,ymax+50)

plt.xlim(pd.to_datetime(i0)[0],pd.to_datetime(i0)[-1])
plt.title(u"%s~2016-11-20の1週間"%day.strftime('%Y-%m-%d'),size='22',
          fontproperties=prop)  #日付変更
plt.xlabel('time', size='22')
plt.ylabel(u'Heart ( 回/min )', size='20',fontproperties=prop)
plt.show()
```

A.4 analysis_month.py

```
#!/Usr/bin/env python
# -*- coding: utf-8 -*-

""" Fitbit """
import fitbit
import gather_keys_oauth2 as Oauth2
import matplotlib.pyplot as plt
import numpy as np
```

```
""" Gmail """
import os.path
import datetime
import smtplib
from email import Encoders
from email.Utils import formatdate
from email.MIMEBase import MIMEBase
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText

import time
import pandas as pd
import matplotlib.font_manager as fm
import os

""" for OAuth2.0 """
USER_ID = ''          #Charge HR の USER_ID 入力
CLIENT_SECRET = ''    # CLIENT_SECRET 入力

""" for obtaining Access-token and Refresh-token """
server = Oauth2.OAuth2Server(USER_ID, CLIENT_SECRET)
server.browser_authorize()

print('FULL RESULTS = %s' % server.oauth.token)
print('ACCESS_TOKEN = %s' % server.oauth.token['access_token'])

ACCESS_TOKEN = server.oauth.token['access_token']
REFRESH_TOKEN = server.oauth.token['refresh_token']

""" Authorization """
auth2_client = fitbit.Fitbit(USER_ID, CLIENT_SECRET, oauth2=True,
access_token=ACCESS_TOKEN, refresh_token=REFRESH_TOKEN)

day = datetime.datetime.strptime('2016-10-01','%Y-%m-%d') #取得する日付に変更
D0 =  datetime.datetime.strptime('2016-10-01','%Y-%m-%d') #取得する日付に変更
""" Getting data """
fitbit_stats = auth2_client.intraday_time_series('activities/heart', day,
              detail_level='1min',start_time="09:00",end_time='18:00')
#開始時刻と終了時刻に変更

""" Getting only 'heartrate' and 'time' """
stats = fitbit_stats['activities-heart-intraday']['dataset']
```

```
txt_name = "{:%Y%m%d}.txt".format(day)

"""time"""
for i in range(1,31):    #１ヶ月の例です。３ヶ月の場合 for i in range(1,92):
    locals()["D%d" % i ]=day + datetime.timedelta(days=i)
print D30



f999 = open(txt_name, 'w')
HR = []
for var in range(0,len(stats)):
    f999.write(str(stats[var]['value']) )
    f999.write("\t")
    f999.write(stats[var]['time'])
    f999.write("\n")
    HR = HR + [stats[var]['value']]
f999.close()



"""時系列グラフ表示"""
plt.style.use('ggplot')
font = {'family' : 'meiryo'}
plt.rc('font', **font)

i0=[]
j0=[]
l=[]
m=[]
for ii in range(1,31): #１ヶ月の例です。３ヶ月の場合 for i in range(1,92):
    locals()["i%d" % ii ]=[]
    locals()["f%d" % ii ] = open("{:%Y%m%d}-time.txt".format
                                 (locals()["D%d" % ii ]), 'r')
    for data in open("{:%Y%m%d}-time.txt".format(locals()["D%d" % ii ]), 'r'):
        heart = data.split('\n')
        locals()["f%d" % ii ].close()
        locals()["h%d" % ii ] = heart[0].split('\n')
        locals()["i%d" % ii ]=locals()["i%d" % ii ]+locals()["h%d" % ii ]

f1 = open("{:%Y%m%d}-time.txt".format(day), 'r')
for data in open("{:%Y%m%d}-time.txt".format(day), 'r'):
    heart = data.split('\n')
    f1.close()
    h = heart[0].split('\n')
```

```
    i0=i0+h


for i in range(1,31):  #１ヶ月の例です。３ヶ月の場合 for i in range(1,92):
    locals()["j%d" % i ]=[]
    locals()["ff%d" % i ] = open("{:%Y%m%d}-heart.txt".format
                                (locals()["D%d" % i ]), 'r')
    for data in open("{:%Y%m%d}-heart.txt".format(locals()["D%d" % i ]), 'r'):
        time = data.split('\n')
        locals()["ff%d" % i ].close()
        locals()["t%d" % i ] = time[0].split('\n')
        locals()["j%d" % i ]=locals()["j%d" % i ]+locals()["t%d" % i ]

ff1 = open("{:%Y%m%d}-heart.txt".format(day), 'r')
for data in open("{:%Y%m%d}-heart.txt".format(day), 'r'):
    time = data.split('\n')
    ff1.close()
    t = time[0].split('\n')
    j0=j0+t




print "----------------------------------------"
print day.strftime('%Y-%m-%d'),"?%s"%D30.strftime('%Y-%m-%d')
print "----------------------------------------"


for k in range(len(locals()["i%d" % ii ])):
    k=[40]
    l=l+k
for k in range(len(locals()["i%d" % ii ])):
    k=[150]
    m=m+k

prop = fm.FontProperties(fname='/usr/share/fonts/ja/TrueType/
                           kochi-gothic-subst.ttf')
plt.plot(pd.to_datetime(locals()["i%d" % ii ]),m,"k",label = "150")
plt.plot(pd.to_datetime(locals()["i%d" % ii ]),l,"k",label = "40")

for i in range(0,6):
    if locals()["D%d" % i ].isoweekday()==1:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
```

```
        "green",label = "Mon")
    if locals()["D%d" % i ].isoweekday()==2:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "orangered",label = "Tue")
    if locals()["D%d" % i ].isoweekday()==3:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "b",label = "Wed")
    if locals()["D%d" % i ].isoweekday()==4:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "m",label = "Thu")
    if locals()["D%d" % i ].isoweekday()==5:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "#660000",label = "Fri")
    if locals()["D%d" % i ].isoweekday()==6:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "#00CCCC",label = "Sat")
    if locals()["D%d" % i ].isoweekday()==7:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "#FFCC00",label = "Sun")
plt.legend(fontsize=7)

for i in range(8,31):  #１ヶ月の例です。３ヶ月の場合 for i in range(8,92):
    if locals()["D%d" % i ].isoweekday()==1:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "green")
    if locals()["D%d" % i ].isoweekday()==2:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "orangered")
    if locals()["D%d" % i ].isoweekday()==3:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ]"b")
    if locals()["D%d" % i ].isoweekday()==4:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],"m")
    if locals()["D%d" % i ].isoweekday()==5:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "#660000")
    if locals()["D%d" % i ].isoweekday()==6:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "#00CCCC")
    if locals()["D%d" % i ].isoweekday()==7:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "#FFCC00")
```

```
Ymax=[]
for i in range(1,31): #１ヶ月の例です。３ヶ月の場合 for i in range(1,92):
    locals()["y%d" % i ]=map(int,locals()["j%d" % i ])
    Ymax=Ymax+locals()["y%d" % i ]
ymax=max(Ymax)


if ymax<150:
    h=160
    plt.ylim(0,h)
else:
    plt.ylim(0,ymax+50)


plt.xlim(pd.to_datetime(i0)[0],pd.to_datetime(i0)[-1])
plt.title(u"%s~2016-10-31 の１ヶ月間"%day.strftime('%Y-%m-%d'),size='22',
          fontproperties=prop) #日付変更
plt.xlabel('time', size='22')
plt.ylabel(u'Heart ( 回/min )', size='20',fontproperties=prop)
plt.show()
```

A.5 analysis_6mon.py

```
#!/Usr/bin/env python
# -*- coding: utf-8 -*-

""" Fitbit """
import fitbit
import gather_keys_oauth2 as Oauth2
import matplotlib.pyplot as plt
import numpy as np

""" Gmail """
import os.path
import datetime
import smtplib
from email import Encoders
from email.Utils import formatdate
from email.MIMEBase import MIMEBase
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText
```

```
import time
import pandas as pd
import matplotlib.font_manager as fm
import os

""" for OAuth2.0 """
USER_ID = ''          #Charge HR の USER_ID 入力
CLIENT_SECRET = ''    # CLIENT_SECRET 入力

""" for obtaining Access-token and Refresh-token """
server = Oauth2.OAuth2Server(USER_ID, CLIENT_SECRET)
server.browser_authorize()

print('FULL RESULTS = %s' % server.oauth.token)
print('ACCESS_TOKEN = %s' % server.oauth.token['access_token'])

ACCESS_TOKEN = server.oauth.token['access_token']
REFRESH_TOKEN = server.oauth.token['refresh_token']

""" Authorization """
auth2_client = fitbit.Fitbit(USER_ID, CLIENT_SECRET, oauth2=True,
                access_token=ACCESS_TOKEN, refresh_token=REFRESH_TOKEN)

  day = datetime.datetime.strptime('2016-05-20','%Y-%m-%d') #取得する日付に変更
  D0 =  datetime.datetime.strptime('2016-05-20','%Y-%m-%d') #取得する日付に変更
""" Getting data """

fitbit_stats = auth2_client.intraday_time_series('activities/heart', day,
              detail_level='1min',start_time="09:00",end_time='18:00')
##開始時刻と終了時刻に変更

""" Getting only 'heartrate' and 'time' """
stats = fitbit_stats['activities-heart-intraday']['dataset']


""" Timeseries data of Heartrate """
txt_name = "{:%Y%m%d}.txt".format(day)


"""time"""
for i in range(1,153):
    locals()["D%d" % i ]=day + datetime.timedelta(days=i)
print D152
```

```
f999 = open(txt_name, 'w')
HR = []
for var in range(0,len(stats)):
    f999.write(str(stats[var]['value']) )
    f999.write("\t")
    f999.write(stats[var]['time'])
    f999.write("\n")
    HR = HR + [stats[var]['value']]
f999.close()


"""時系列グラフ表示"""
plt.style.use('ggplot')
font = {'family' : 'meiryo'}
plt.rc('font', **font)

i0=[]
j0=[]
for ii in range(1,153):
    locals()["i%d" % ii ]=[]
    locals()["f%d" % ii ] = open("{:%Y%m%d}-time.txt".format
                            (locals()["D%d" % ii ]), 'r')
    for data in open("{:%Y%m%d}-time.txt".format(locals()["D%d" % ii ]), 'r'):
        heart = data.split('\n')
        locals()["f%d" % ii ].close()
        locals()["h%d" % ii ] = heart[0].split('\n')
        locals()["i%d" % ii ]=locals()["i%d" % ii ]+locals()["h%d" % ii ]

f1 = open("{:%Y%m%d}-time.txt".format(day), 'r')
for data in open("{:%Y%m%d}-time.txt".format(day), 'r'):
    heart = data.split('\n')
    f1.close()
    h = heart[0].split('\n')
    i0=i0+h



for i in range(1,153):
    locals()["j%d" % i ]=[]
    locals()["ff%d" % i ] = open("{:%Y%m%d}-heart.txt".format
                            (locals()["D%d" % i ]), 'r')
```

```
    for data in open("{:%Y%m%d}-heart.txt".format(locals()["D%d" % i ]), 'r'):
        time = data.split('\n')
        locals()["ff%d" % i ].close()
        locals()["t%d" % i ] = time[0].split('\n')
        locals()["j%d" % i ]=locals()["j%d" % i ]+locals()["t%d" % i ]

ff1 = open("{:%Y%m%d}-heart.txt".format(day), 'r')
for data in open("{:%Y%m%d}-heart.txt".format(day), 'r'):
    time = data.split('\n')
    ff1.close()
    t = time[0].split('\n')
    j0=j0+t



print "----------------------------------------"
print day.strftime('%Y-%m-%d'),"?%s"%D152.strftime('%Y-%m-%d')
print "----------------------------------------"

m=[]
l=[]
for k in range(len(locals()["i%d" % ii ])):
    k=[40]
    l=l+k
for k in range(len(locals()["i%d" % ii ])):
    k=[150]
    m=m+k

prop = fm.FontProperties(fname='/usr/share/fonts/ja/TrueType/
                          kochi-gothic-subst.ttf')
plt.plot(pd.to_datetime(locals()["i%d" % ii ]),m,"k",label = "150")
plt.plot(pd.to_datetime(locals()["i%d" % ii ]),l,"k",label = "40")


for i in range(0,6):
  if locals()["D%d" % i ].isoweekday()==1:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "green",label = "Mon")
#  if locals()["D%d" % i ].isoweekday()==2:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
#        "orangered",label = "Tue")
#  if locals()["D%d" % i ].isoweekday()==3:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
```

```
#        "b",label = "Wed")
#  if locals()["D%d" % i ].isoweekday()==4:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
#        "m",label = "Thu")
#  if locals()["D%d" % i ].isoweekday()==5:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
#        "#660000",label = "Fri")
#  if locals()["D%d" % i ].isoweekday()==6:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
#        "#00CCCC",label = "Sat")
#  if locals()["D%d" % i ].isoweekday()==7:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
#        "#FFCC00",label = "Sun")
  plt.legend(fontsize=7)
```

#上は月曜日の例です。
#曜日毎に＃を外してください。

```
for i in range(8,153):
    if locals()["D%d" % i ].isoweekday()==1:
        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
        "green")
#    if locals()["D%d" % i ].isoweekday()==2:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
#        "orangered")
#    if locals()["D%d" % i ].isoweekday()==3:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],"b")
#    if locals()["D%d" % i ].isoweekday()==4:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],"m")
#    if locals()["D%d" % i ].isoweekday()==5:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
#        "#660000")
#    if locals()["D%d" % i ].isoweekday()==6:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
#        "#00CCCC")
#    if locals()["D%d" % i ].isoweekday()==7:
#        plt.plot(pd.to_datetime(locals()["i%d" % i ]),locals()["j%d" % i ],
#        "#FFCC00")
```

#上は月曜日の例です。
#曜日毎に＃を外してください。

```
Ymax=[]
for i in range(1,153):
    locals()["y%d" % i ]=map(int,locals()["j%d" % i ])
    Ymax=Ymax+locals()["y%d" % i ]
ymax=max(Ymax)


if ymax<150:
    h=160
    plt.ylim(0,h)
else:
    plt.ylim(0,ymax+20)

plt.xlim(pd.to_datetime(i0)[0],pd.to_datetime(i0)[-1])
plt.title(u"%s~2016-10-19 の 6 ヶ月間の月曜日"%day.strftime('%Y-%m-%d'),size='22',
          fontproperties=prop) #日付変更
plt.xlabel('time', size='22')
plt.ylabel(u'Heart ( 回/min )', size='20',fontproperties=prop)
plt.show()
plt.show()
```

付録 B　関連ソースコード

B.1　____ init ____.py

```
# -*- coding: utf-8 -*-
"""
Fitbit API Library
------------------
:copyright: 2012-2015 ORCAS.

:license: BSD, see LICENSE for more details.
"""

from .api import Fitbit, FitbitOauth2Client

# Meta.

__title__ = 'fitbit'
__author__ = 'Issac Kelly and ORCAS'
__author_email__ = 'bpitcher@orcasinc.com'
```

```
__copyright__ = 'Copyright 2012-2015 ORCAS'
__license__ = 'Apache 2.0'

__version__ = '0.2.3'
__release__ = '0.2.3'

# Module namespace.

all_tests = []
```

B.2 api.py

```
# -*- coding: utf-8 -*-
import datetime
import json
import requests

try:
    from urllib.parse import urlencode
except ImportError:
    # Python 2.x
    from urllib import urlencode

from requests_oauthlib import OAuth2, OAuth2Session
from oauthlib.oauth2.rfc6749.errors import TokenExpiredError
from fitbit.exceptions import (BadResponse, DeleteError, HTTPBadRequest,
                               HTTPUnauthorized, HTTPForbidden,
                               HTTPServerError, HTTPConflict, HTTPNotFound,
                               HTTPTooManyRequests)
from fitbit.utils import curry


class FitbitOauth2Client(object):
    API_ENDPOINT = "https://api.fitbit.com"
    AUTHORIZE_ENDPOINT = "https://www.fitbit.com"
    API_VERSION = 1

    request_token_url = "%s/oauth2/token" % API_ENDPOINT
    authorization_url = "%s/oauth2/authorize" % AUTHORIZE_ENDPOINT
    access_token_url = request_token_url
    refresh_token_url = request_token_url

    def __init__(self, client_id, client_secret,
```

```
            access_token=None, refresh_token=None,
            *args, **kwargs):
    """
    Create a FitbitOauth2Client object. Specify the first 7 parameters if
    you have them to access user data. Specify just the first 2 parameters
    to start the setup for user authorization (as an example see gather_
    key_oauth2.py)
        - client_id, client_secret are in the app configuration page
        https://dev.fitbit.com/apps
        - access_token, refresh_token are obtained after the user grants
        permission
    """

    self.session = requests.Session()
    self.client_id = client_id
    self.client_secret = client_secret
    self.token = {
        'access_token': access_token,
        'refresh_token': refresh_token
    }
    self.oauth = OAuth2Session(client_id)

def _request(self, method, url, **kwargs):
    """
    A simple wrapper around requests.
    """
    return self.session.request(method, url, **kwargs)

def make_request(self, url, data={}, method=None, **kwargs):
    """
    Builds and makes the OAuth2 Request, catches errors
    https://wiki.fitbit.com/display/API/API+Response+Format+And+Errors
    """
    if not method:
        method = 'POST' if data else 'GET'

    try:
        auth = OAuth2(client_id=self.client_id, token=self.token)
        response = self._request(method, url, data=data, auth=auth, **kwargs)
    except (HTTPUnauthorized, TokenExpiredError) as e:
        self.refresh_token()
        auth = OAuth2(client_id=self.client_id, token=self.token)
        response = self._request(method, url, data=data, auth=auth, **kwargs)
```

```
    # yet another token expiration check
    # (the above try/except only applies if the expired token was obtained
    # using the current instance of the class this is a a general case)
    if response.status_code == 401:
        d = json.loads(response.content.decode('utf8'))
        try:
            if(d['errors'][0]['errorType'] == 'expired_token' and
                d['errors'][0]['message'].find('Access token expired:')==0):
                    self.refresh_token()
                    auth = OAuth2(client_id=self.client_id, token=self.token)
                    response = self._request(method, url, data=data,
                    auth=auth, **kwargs)
        except:
            pass

    if response.status_code == 401:
        raise HTTPUnauthorized(response)
    elif response.status_code == 403:
        raise HTTPForbidden(response)
    elif response.status_code == 404:
        raise HTTPNotFound(response)
    elif response.status_code == 409:
        raise HTTPConflict(response)
    elif response.status_code == 429:
        exc = HTTPTooManyRequests(response)
        exc.retry_after_secs = int(response.headers['Retry-After'])
        raise exc

    elif response.status_code >= 500:
        raise HTTPServerError(response)
    elif response.status_code >= 400:
        raise HTTPBadRequest(response)
    return response

def authorize_token_url(self, scope=None, redirect_uri=None, **kwargs):
    """Step 1: Return the URL the user needs to go to in order to grant us
    authorization to look at their data.  Then redirect the user to that
    URL, open their browser to it, or tell them to copy the URL into their
    browser.
        - scope: pemissions that that are being requested [default ask all]
        - redirect_uri: url to which the reponse will posted
                        required only if your app does not have one
```

```
        for more info see https://wiki.fitbit.com/display/API/OAuth+2.0
    """

    # the scope parameter is caussing some issues when refreshing tokens
    # so not saving it
    old_scope = self.oauth.scope
    old_redirect = self.oauth.redirect_uri
    if scope:
        self.oauth.scope = scope
    else:
        self.oauth.scope = [
            "activity", "nutrition", "heartrate", "location", "nutrition",
            "profile", "settings", "sleep", "social", "weight"
        ]

    if redirect_uri:
        self.oauth.redirect_uri = redirect_uri

    out = self.oauth.authorization_url(self.authorization_url, **kwargs)
    self.oauth.scope = old_scope
    self.oauth.redirect_uri = old_redirect
    return(out)

def fetch_access_token(self, code, redirect_uri):

    """Step 2: Given the code from fitbit from step 1, call
    fitbit again and returns an access token object. Extract the needed
    information from that and save it to use in future API calls.
    the token is internally saved
    """
    auth = OAuth2Session(self.client_id, redirect_uri=redirect_uri)
    self.token = auth.fetch_token(
        self.access_token_url,
        username=self.client_id,
        password=self.client_secret,
        code=code)

    return self.token

def refresh_token(self):
    """Step 3: obtains a new access_token from the the refresh token
    obtained in step 2.
    the token is internally saved
```

```
        """
        self.token = self.oauth.refresh_token(
            self.refresh_token_url,
            refresh_token=self.token['refresh_token'],
            auth=requests.auth.HTTPBasicAuth(self.client_id, self.client_secret)
        )

        return self.token


class Fitbit(object):
    US = 'en_US'
    METRIC = 'en_UK'

    API_ENDPOINT = "https://api.fitbit.com"
    API_VERSION = 1
    WEEK_DAYS = ['SUNDAY', 'MONDAY', 'TUESDAY', 'WEDNESDAY', 'THURSDAY',
                 'FRIDAY', 'SATURDAY']
    PERIODS = ['1d', '7d', '30d', '1w', '1m', '3m', '6m', '1y', 'max']

    RESOURCE_LIST = [
        'body',
        'activities',
        'foods/log',
        'foods/log/water',
        'sleep',
        'heart',
        'bp',
        'glucose',
    ]

    QUALIFIERS = [
        'recent',
        'favorite',
        'frequent',
    ]

    def __init__(self, client_id, client_secret, system=US, **kwargs):
        """
        Fitbit(<id>, <secret>, access_token=<token>, refresh_token=<token>)
        """
        self.system = system
        self.client = FitbitOauth2Client(client_id, client_secret, **kwargs)
```

```
    # All of these use the same patterns, define the method for accessing
    # creating and deleting records once, and use curry to make individual
    # Methods for each
    for resource in Fitbit.RESOURCE_LIST:
        underscore_resource = resource.replace('/', '_')
        setattr(self, underscore_resource,
                curry(self._COLLECTION_RESOURCE, resource))

        if resource not in ['body', 'glucose']:
            # Body and Glucose entries are not currently able to be deleted
            setattr(self, 'delete_%s' % underscore_resource, curry(
                self._DELETE_COLLECTION_RESOURCE, resource))

    for qualifier in Fitbit.QUALIFIERS:
        setattr(self, '%s_activities' % qualifier,
                curry(self.activity_stats, qualifier=qualifier))
        setattr(self, '%s_foods' % qualifier, curry(self._food_stats,
                                                    qualifier=qualifier))

def make_request(self, *args, **kwargs):
    # This should handle data level errors, improper requests, and bad
    # serialization
    headers = kwargs.get('headers', {})
    headers.update({'Accept-Language': self.system})
    kwargs['headers'] = headers

    method = kwargs.get('method', 'POST' if 'data' in kwargs else 'GET')
    response = self.client.make_request(*args, **kwargs)

    if response.status_code == 202:
        return True
    if method == 'DELETE':
        if response.status_code == 204:
            return True
        else:
            raise DeleteError(response)
    try:
        rep = json.loads(response.content.decode('utf8'))
    except ValueError:
        raise BadResponse

    return rep
```

```
def user_profile_get(self, user_id=None):
    """
    Get a user profile. You can get other user's profile information
    by passing user_id, or you can get the current user's by not passing
    a user_id
    .. note:
        This is not the same format that the GET comes back in, GET requests
        are wrapped in {'user': <dict of user data>}
    https://wiki.fitbit.com/display/API/API-Get-User-Info
    """
    url = "{0}/{1}/user/{2}/profile.json".format
                (*self._get_common_args(user_id))
    return self.make_request(url)

def user_profile_update(self, data):
    """
    Set a user profile. You can set your user profile information by
    passing a dictionary of attributes that will be updated.
    .. note:
        This is not the same format that the GET comes back in, GET requests
        are wrapped in {'user': <dict of user data>}
    https://wiki.fitbit.com/display/API/API-Update-User-Info
    """
    url = "{0}/{1}/user/-/profile.json".format(*self._get_common_args())
    return self.make_request(url, data)

def _get_common_args(self, user_id=None):
    common_args = (self.API_ENDPOINT, self.API_VERSION,)
    if not user_id:
        user_id = '-'
    common_args += (user_id,)
    return common_args

def _get_date_string(self, date):
    if not isinstance(date, str):
        return date.strftime('%Y-%m-%d')
    return date

def _COLLECTION_RESOURCE(self, resource, date=None, user_id=None,
                         data=None):
    """
    Retrieving and logging of each type of collection data.
```

```
    Arguments:
        resource, defined automatically via curry
        [date] defaults to today
        [user_id] defaults to current logged in user
        [data] optional, include for creating a record, exclude for access
    This implements the following methods::
        body(date=None, user_id=None, data=None)
        activities(date=None, user_id=None, data=None)
        foods_log(date=None, user_id=None, data=None)
        foods_log_water(date=None, user_id=None, data=None)
        sleep(date=None, user_id=None, data=None)
        heart(date=None, user_id=None, data=None)
        bp(date=None, user_id=None, data=None)
    * https://wiki.fitbit.com/display/API/Fitbit+Resource+Access+API
    """

    if not date:
        date = datetime.date.today()
    date_string = self._get_date_string(date)

    kwargs = {'resource': resource, 'date': date_string}
    if not data:
        base_url = "{0}/{1}/user/{2}/{resource}/date/{date}.json"
    else:
        data['date'] = date_string
        base_url = "{0}/{1}/user/{2}/{resource}.json"
    url = base_url.format(*self._get_common_args(user_id), **kwargs)
    return self.make_request(url, data)

def _DELETE_COLLECTION_RESOURCE(self, resource, log_id):
    """
    deleting each type of collection data
    Arguments:
        resource, defined automatically via curry
        log_id, required, log entry to delete
    This builds the following methods::
        delete_body(log_id)
        delete_activities(log_id)
        delete_foods_log(log_id)
        delete_foods_log_water(log_id)
        delete_sleep(log_id)
        delete_heart(log_id)
        delete_bp(log_id)
```

```
    """
    url = "{0}/{1}/user/-/{resource}/{log_id}.json".format(
        *self._get_common_args(),
        resource=resource,
        log_id=log_id
    )
    response = self.make_request(url, method='DELETE')
    return response

def _resource_goal(self, resource, data={}, period=None):
    """ Handles GETting and POSTing resource goals of all types """
    url = "{0}/{1}/user/-/{resource}/goal{postfix}.json".format(
        *self._get_common_args(),
        resource=resource,
        postfix=('s/' + period) if period else ''
    )
    return self.make_request(url, data=data)

def _filter_nones(self, data):
    filter_nones = lambda item: item[1] is not None
    filtered_kwargs = list(filter(filter_nones, data.items()))
    return {} if not filtered_kwargs else dict(filtered_kwargs)

def body_fat_goal(self, fat=None):
    """
    Implements the following APIs
    * https://wiki.fitbit.com/display/API/API-Get-Body-Fat
    * https://wiki.fitbit.com/display/API/API-Update-Fat-Goal
    Pass no arguments to get the body fat goal. Pass ``fat`` argument
    to update the body fat goal.
    Arguments:
    * ``fat`` -- Target body fat in %; in the format X.XX
    """
    return self._resource_goal('body/log/fat', {'fat': fat} if fat else {})

def body_weight_goal(self, start_date=None, start_weight=None, weight=None):
    """
    Implements the following APIs
    * https://wiki.fitbit.com/display/API/API-Get-Body-Weight-Goal
    * https://wiki.fitbit.com/display/API/API-Update-Weight-Goal
    Pass no arguments to get the body weight goal. Pass ``start_date``,
    ``start_weight`` and optionally ``weight`` to set the weight goal.
    ``weight`` is required if it hasn't been set yet.
```

```
    Arguments:
    * ``start_date`` -- Weight goal start date; in the format yyyy-MM-dd
    * ``start_weight`` -- Weight goal start weight; in the format X.XX
    * ``weight`` -- Weight goal target weight; in the format X.XX
    """
    data = self._filter_nones({
        'startDate': start_date,
        'startWeight': start_weight,
        'weight': weight
    })
    if data and not ('startDate' in data and 'startWeight' in data):
        raise ValueError('start_date and start_weight are both required')
    return self._resource_goal('body/log/weight', data)

def activities_daily_goal(self, calories_out=None, active_minutes=None,
                          floors=None, distance=None, steps=None):
    """
    Implements the following APIs
    https://wiki.fitbit.com/display/API/API-Get-Activity-Daily-Goals
    https://wiki.fitbit.com/display/API/API-Update-Activity-Daily-Goals
    Pass no arguments to get the daily activities goal. Pass any one of
    the optional arguments to set that component of the daily activities
    goal.
    Arguments:
    * ``calories_out`` -- New goal value; in an integer format
    * ``active_minutes`` -- New goal value; in an integer format
    * ``floors`` -- New goal value; in an integer format
    * ``distance`` -- New goal value; in the format X.XX or integer
    * ``steps`` -- New goal value; in an integer format
    """
    data = self._filter_nones({
        'caloriesOut': calories_out,
        'activeMinutes': active_minutes,
        'floors': floors,
        'distance': distance,
        'steps': steps
    })
    return self._resource_goal('activities', data, period='daily')

def activities_weekly_goal(self, distance=None, floors=None, steps=None):
    """
    Implements the following APIs
    https://wiki.fitbit.com/display/API/API-Get-Activity-Weekly-Goals
```

```
    https://wiki.fitbit.com/display/API/API-Update-Activity-Weekly-Goals
    Pass no arguments to get the weekly activities goal. Pass any one of
    the optional arguments to set that component of the weekly activities
    goal.
    Arguments:
    * ``distance`` -- New goal value; in the format X.XX or integer
    * ``floors`` -- New goal value; in an integer format
    * ``steps`` -- New goal value; in an integer format
    """
    data = self._filter_nones({'distance': distance, 'floors': floors,
                               'steps': steps})
    return self._resource_goal('activities', data, period='weekly')

def food_goal(self, calories=None, intensity=None, personalized=None):
    """
    Implements the following APIs
    https://wiki.fitbit.com/display/API/API-Get-Food-Goals
    https://wiki.fitbit.com/display/API/API-Update-Food-Goals
    Pass no arguments to get the food goal. Pass at least ``calories`` or
    ``intensity`` and optionally ``personalized`` to update the food goal.
    Arguments:
    * ``calories`` -- Manual Calorie Consumption Goal; calories, integer;
    * ``intensity`` -- Food Plan intensity; (MAINTENANCE, EASIER, MEDIUM,
                       KINDAHARD, HARDER);
    * ``personalized`` -- Food Plan type; ``True`` or ``False``
    """
    data = self._filter_nones({'calories': calories, 'intensity': intensity,
                               'personalized': personalized})
    if data and not ('calories' in data or 'intensity' in data):
        raise ValueError('Either calories or intensity is required')
    return self._resource_goal('foods/log', data)

def water_goal(self, target=None):
    """
    Implements the following APIs
    https://wiki.fitbit.com/display/API/API-Get-Water-Goal
    https://wiki.fitbit.com/display/API/API-Update-Water-Goal
    Pass no arguments to get the water goal. Pass ``target`` to update it.
    Arguments:
    * ``target`` -- Target water goal in the format X.X, will be set in
                    unit based on locale
    """
    data = self._filter_nones({'target': target})
```

```
    return self._resource_goal('foods/log/water', data)

def time_series(self, resource, user_id=None, base_date='today',
                period=None, end_date=None):
    """
    The time series is a LOT of methods, (documented at url below) so they
    don't get their own method. They all follow the same patterns, and
    return similar formats.
    Taking liberty, this assumes a base_date of today, the current user,
    and a 1d period.
    https://wiki.fitbit.com/display/API/API-Get-Time-Series
    """
    if period and end_date:
        raise TypeError("Either end_date or period can be specified,
                        not both")

    if end_date:
        end = self._get_date_string(end_date)
    else:
        if not period in Fitbit.PERIODS:
            raise ValueError("Period must be one of %s"
                             % ','.join(Fitbit.PERIODS))
        end = period

    url = "{0}/{1}/user/{2}/{resource}/date/{base_date}/{end}.json".format(
        *self._get_common_args(user_id),
        resource=resource,
        base_date=self._get_date_string(base_date),
        end=end
    )
    return self.make_request(url)

def intraday_time_series(self, resource, base_date='today',
detail_level='1min', start_time=None, end_time=None):
    """
    The intraday time series extends the functionality of the regular time
    series, but returning data at a
    more granular level for a single day, defaulting to 1 minute intervals.
    To access this feature, one must
    send an email to api@fitbit.com and request to have access to
    the Partner API
    (see https://wiki.fitbit.com/display/API/Fitbit+Partner+API).
    For details on the resources available, see:
```

```
    https://wiki.fitbit.com/display/API/API-Get-Intraday-Time-Series
    """

    # Check that the time range is valid
    time_test = lambda t: not (t is None or isinstance(t, str) and not t)
    time_map = list(map(time_test, [start_time, end_time]))
    if not all(time_map) and any(time_map):
        raise TypeError('You must provide both the end and start time
        or neither')

    """
    Per
    https://wiki.fitbit.com/display/API/API-Get-Intraday-Time-Series
    the detail-level is now (OAuth 2.0 ):
    either "1min" or "15min" (optional). "1sec" for heart rate.
    """
    if not detail_level in ['1sec', '1min', '15min']:
        raise ValueError("Period must be either '1sec', '1min', or '15min'")

    url = "{0}/{1}/user/-/{resource}/date/{base_date}/1d/{detail_level}".
    format(
        *self._get_common_args(),
        resource=resource,
        base_date=self._get_date_string(base_date),
        detail_level=detail_level
    )

    if all(time_map):
        url = url + '/time'
        for time in [start_time, end_time]:
            time_str = time
            if not isinstance(time_str, str):
                time_str = time.strftime('%H:%M')
            url = url + ('/%s' % (time_str))

    url = url + '.json'

    return self.make_request(url)

def activity_stats(self, user_id=None, qualifier=''):
    """
    * https://wiki.fitbit.com/display/API/API-Get-Activity-Stats
    * https://wiki.fitbit.com/display/API/API-Get-Favorite-Activities
```

```
    * https://wiki.fitbit.com/display/API/API-Get-Recent-Activities
    * https://wiki.fitbit.com/display/API/API-Get-Frequent-Activities
    This implements the following methods::
        recent_activities(user_id=None, qualifier='')
        favorite_activities(user_id=None, qualifier='')
        frequent_activities(user_id=None, qualifier='')
    """
    if qualifier:
        if qualifier in Fitbit.QUALIFIERS:
            qualifier = '/%s' % qualifier
        else:
            raise ValueError("Qualifier must be one of %s"
                             % ', '.join(Fitbit.QUALIFIERS))
    else:
        qualifier = ''

    url = "{0}/{1}/user/{2}/activities{qualifier}.json".format(
        *self._get_common_args(user_id),
        qualifier=qualifier
    )
    return self.make_request(url)

def _food_stats(self, user_id=None, qualifier=''):
    """
    This builds the convenience methods on initialization::
        recent_foods(user_id=None, qualifier='')
        favorite_foods(user_id=None, qualifier='')
        frequent_foods(user_id=None, qualifier='')
    * https://wiki.fitbit.com/display/API/API-Get-Recent-Foods
    * https://wiki.fitbit.com/display/API/API-Get-Frequent-Foods
    * https://wiki.fitbit.com/display/API/API-Get-Favorite-Foods
    """
    url = "{0}/{1}/user/{2}/foods/log/{qualifier}.json".format(
        *self._get_common_args(user_id),
        qualifier=qualifier
    )
    return self.make_request(url)

def add_favorite_activity(self, activity_id):
    """
    https://wiki.fitbit.com/display/API/API-Add-Favorite-Activity
    """
    url = "{0}/{1}/user/-/activities/favorite/{activity_id}.json".format(
```

```
        *self._get_common_args(),
        activity_id=activity_id
    )
    return self.make_request(url, method='POST')

def log_activity(self, data):
    """
    https://wiki.fitbit.com/display/API/API-Log-Activity
    """
    url = "{0}/{1}/user/-/activities.json".format(*self._get_common_args())
    return self.make_request(url, data=data)

def delete_favorite_activity(self, activity_id):
    """
    https://wiki.fitbit.com/display/API/API-Delete-Favorite-Activity
    """
    url = "{0}/{1}/user/-/activities/favorite/{activity_id}.json".format(
        *self._get_common_args(),
        activity_id=activity_id
    )
    return self.make_request(url, method='DELETE')

def add_favorite_food(self, food_id):
    """
    https://wiki.fitbit.com/display/API/API-Add-Favorite-Food
    """
    url = "{0}/{1}/user/-/foods/log/favorite/{food_id}.json".format(
        *self._get_common_args(),
        food_id=food_id
    )
    return self.make_request(url, method='POST')

def delete_favorite_food(self, food_id):
    """
    https://wiki.fitbit.com/display/API/API-Delete-Favorite-Food
    """
    url = "{0}/{1}/user/-/foods/log/favorite/{food_id}.json".format(
        *self._get_common_args(),
        food_id=food_id
    )
    return self.make_request(url, method='DELETE')

def create_food(self, data):
```

```
    """
    https://wiki.fitbit.com/display/API/API-Create-Food
    """
    url = "{0}/{1}/user/-/foods.json".format(*self._get_common_args())
    return self.make_request(url, data=data)

def get_meals(self):
    """
    https://wiki.fitbit.com/display/API/API-Get-Meals
    """
    url = "{0}/{1}/user/-/meals.json".format(*self._get_common_args())
    return self.make_request(url)

def get_devices(self):
    """
    https://wiki.fitbit.com/display/API/API-Get-Devices
    """
    url = "{0}/{1}/user/-/devices.json".format(*self._get_common_args())
    return self.make_request(url)

def get_alarms(self, device_id):
    """
    https://wiki.fitbit.com/display/API/API-Devices-Get-Alarms
    """
    url = "{0}/{1}/user/-/devices/tracker/{device_id}/alarms.json".format(
        *self._get_common_args(),
        device_id=device_id
    )
    return self.make_request(url)

def add_alarm(self, device_id, alarm_time, week_days, recurring=False,
              enabled=True, label=None, snooze_length=None,
              snooze_count=None, vibe='DEFAULT'):
    """
    https://wiki.fitbit.com/display/API/API-Devices-Add-Alarm
    alarm_time should be a timezone aware datetime object.
    """
    url = "{0}/{1}/user/-/devices/tracker/{device_id}/alarms.json".format(
        *self._get_common_args(),
        device_id=device_id
    )
    alarm_time = alarm_time.strftime("%H:%M%z")
    # Check week_days list
```

```
    if not isinstance(week_days, list):
        raise ValueError("Week days needs to be a list")
    for day in week_days:
        if day not in self.WEEK_DAYS:
            raise ValueError("Incorrect week day %s. see WEEK_DAY_LIST."
             % day)
    data = {
        'time': alarm_time,
        'weekDays': week_days,
        'recurring': recurring,
        'enabled': enabled,
        'vibe': vibe
    }
    if label:
        data['label'] = label
    if snooze_length:
        data['snoozeLength'] = snooze_length
    if snooze_count:
        data['snoozeCount'] = snooze_count
    return self.make_request(url, data=data, method="POST")
    # return

def update_alarm(self, device_id, alarm_id, alarm_time, week_days,
recurring=False, enabled=True, label=None,
                 snooze_length=None, snooze_count=None, vibe='DEFAULT'):
    """
    https://wiki.fitbit.com/display/API/API-Devices-Update-Alarm
    alarm_time should be a timezone aware datetime object.
    """
    # TODO Refactor with create_alarm. Tons of overlap.
    # Check week_days list
    if not isinstance(week_days, list):
        raise ValueError("Week days needs to be a list")
    for day in week_days:
        if day not in self.WEEK_DAYS:
            raise ValueError("Incorrect week day %s. see WEEK_DAY_LIST."
            % day)
    url = "{0}/{1}/user/-/devices/tracker/{device_id}/alarms/{alarm_id}.
    json".format(
        *self._get_common_args(),
        device_id=device_id,
        alarm_id=alarm_id
    )
```

```
        alarm_time = alarm_time.strftime("%H:%M%z")

        data = {
            'time': alarm_time,
            'weekDays': week_days,
            'recurring': recurring,
            'enabled': enabled,
            'vibe': vibe
        }
        if label:
            data['label'] = label
        if snooze_length:
            data['snoozeLength'] = snooze_length
        if snooze_count:
            data['snoozeCount'] = snooze_count
        return self.make_request(url, data=data, method="POST")
        # return

    def delete_alarm(self, device_id, alarm_id):
        """
        https://wiki.fitbit.com/display/API/API-Devices-Delete-Alarm
        """
        url = "{0}/{1}/user/-/devices/tracker/{device_id}/alarms/{alarm_id}.
        json".format(
            *self._get_common_args(),
            device_id=device_id,
            alarm_id=alarm_id
        )
        return self.make_request(url, method="DELETE")

    def get_sleep(self, date):
        """
        https://wiki.fitbit.com/display/API/API-Get-Sleep
        date should be a datetime.date object.
        """
        url = "{0}/{1}/user/-/sleep/date/{year}-{month}-{day}.json".format(
            *self._get_common_args(),
            year=date.year,
            month=date.month,
            day=date.day
        )
        return self.make_request(url)
```

```
def log_sleep(self, start_time, duration):
    """
    https://wiki.fitbit.com/display/API/API-Log-Sleep
    start time should be a datetime object. We will be using the year,
    month, day, hour, and minute.
    """
    data = {
        'startTime': start_time.strftime("%H:%M"),
        'duration': duration,
        'date': start_time.strftime("%Y-%m-%d"),
    }
    url = "{0}/{1}/user/-/sleep.json".format(*self._get_common_args())
    return self.make_request(url, data=data, method="POST")

def activities_list(self):
    """
    https://wiki.fitbit.com/display/API/API-Browse-Activities
    """
    url = "{0}/{1}/activities.json".format(*self._get_common_args())
    return self.make_request(url)

def activity_detail(self, activity_id):
    """
    https://wiki.fitbit.com/display/API/API-Get-Activity
    """
    url = "{0}/{1}/activities/{activity_id}.json".format(
        *self._get_common_args(),
        activity_id=activity_id
    )
    return self.make_request(url)

def search_foods(self, query):
    """
    https://wiki.fitbit.com/display/API/API-Search-Foods
    """
    url = "{0}/{1}/foods/search.json?{encoded_query}".format(
        *self._get_common_args(),
        encoded_query=urlencode({'query': query})
    )
    return self.make_request(url)

def food_detail(self, food_id):
    """
```

```
    https://wiki.fitbit.com/display/API/API-Get-Food
    """
    url = "{0}/{1}/foods/{food_id}.json".format(
        *self._get_common_args(),
        food_id=food_id
    )
    return self.make_request(url)

def food_units(self):
    """
    https://wiki.fitbit.com/display/API/API-Get-Food-Units
    """
    url = "{0}/{1}/foods/units.json".format(*self._get_common_args())
    return self.make_request(url)

def get_bodyweight(self, base_date=None, user_id=None, period=None,
 end_date=None):
    """
    https://wiki.fitbit.com/display/API/API-Get-Body-Weight
    base_date should be a datetime.date object (defaults to today),
    period can be '1d', '7d', '30d', '1w', '1m', '3m', '6m', '1y', 'max'
    or None
    end_date should be a datetime.date object, or None.
    You can specify period or end_date, or neither, but not both.
    """
    return self._get_body('weight', base_date, user_id, period, end_date)

def get_bodyfat(self, base_date=None, user_id=None, period=None,
end_date=None):
    """
    https://wiki.fitbit.com/display/API/API-Get-Body-fat
    base_date should be a datetime.date object (defaults to today),
    period can be '1d', '7d', '30d', '1w', '1m', '3m', '6m', '1y',
    'max' or None
    end_date should be a datetime.date object, or None.
    You can specify period or end_date, or neither, but not both.
    """
    return self._get_body('fat', base_date, user_id, period, end_date)

def _get_body(self, type_, base_date=None, user_id=None, period=None,
              end_date=None):
    if not base_date:
        base_date = datetime.date.today()
```

```
    if period and end_date:
        raise TypeError("Either end_date or period can be specified,
        not both")

    base_date_string = self._get_date_string(base_date)

    kwargs = {'type_': type_}
    base_url = "{0}/{1}/user/{2}/body/log/{type_}/date/{date_string}.json"
    if period:
        if not period in Fitbit.PERIODS:
            raise ValueError("Period must be one of %s" %
                             ','.join(Fitbit.PERIODS))
        kwargs['date_string'] = '/'.join([base_date_string, period])
    elif end_date:
        end_string = self._get_date_string(end_date)
        kwargs['date_string'] = '/'.join([base_date_string, end_string])
    else:
        kwargs['date_string'] = base_date_string

    url = base_url.format(*self._get_common_args(user_id), **kwargs)
    return self.make_request(url)

def get_friends(self, user_id=None):
    """
    https://wiki.fitbit.com/display/API/API-Get-Friends
    """
    url = "{0}/{1}/user/{2}/friends.json".format
    (*self._get_common_args(user_id))
    return self.make_request(url)

def get_friends_leaderboard(self, period):
    """
    https://wiki.fitbit.com/display/API/API-Get-Friends-Leaderboard
    """
    if not period in ['7d', '30d']:
        raise ValueError("Period must be one of '7d', '30d'")
    url = "{0}/{1}/user/-/friends/leaders/{period}.json".format(
        *self._get_common_args(),
        period=period
    )
    return self.make_request(url)
```

```
def invite_friend(self, data):
    """
    https://wiki.fitbit.com/display/API/API-Create-Invite
    """
    url = "{0}/{1}/user/-/friends/invitations.json".format
    (*self._get_common_args())
    return self.make_request(url, data=data)

def invite_friend_by_email(self, email):
    """
    Convenience Method for
    https://wiki.fitbit.com/display/API/API-Create-Invite
    """
    return self.invite_friend({'invitedUserEmail': email})

def invite_friend_by_userid(self, user_id):
    """
    Convenience Method for
    https://wiki.fitbit.com/display/API/API-Create-Invite
    """
    return self.invite_friend({'invitedUserId': user_id})

def respond_to_invite(self, other_user_id, accept=True):
    """
    https://wiki.fitbit.com/display/API/API-Accept-Invite
    """
    url = "{0}/{1}/user/-/friends/invitations/{user_id}.json".format(
        *self._get_common_args(),
        user_id=other_user_id
    )
    accept = 'true' if accept else 'false'
    return self.make_request(url, data={'accept': accept})

def accept_invite(self, other_user_id):
    """
    Convenience method for respond_to_invite
    """
    return self.respond_to_invite(other_user_id)

def reject_invite(self, other_user_id):
    """
    Convenience method for respond_to_invite
    """
```

```
        return self.respond_to_invite(other_user_id, accept=False)

    def get_badges(self, user_id=None):
        """
        https://wiki.fitbit.com/display/API/API-Get-Badges
        """
        url = "{0}/{1}/user/{2}/badges.json".format
        (*self._get_common_args(user_id))
        return self.make_request(url)

    def subscription(self, subscription_id, subscriber_id, collection=None,
                     method='POST'):
        """
        https://wiki.fitbit.com/display/API/Fitbit+Subscriptions+API
        """
        base_url = "{0}/{1}/user/-{collection}/apiSubscriptions/
        {end_string}.json"
        kwargs = {'collection': '', 'end_string': subscription_id}
        if collection:
            kwargs = {
                'end_string': '-'.join([subscription_id, collection]),
                'collection': '/' + collection
            }
        return self.make_request(
            base_url.format(*self._get_common_args(), **kwargs),
            method=method,
            headers={"X-Fitbit-Subscriber-id": subscriber_id}
        )

    def list_subscriptions(self, collection=''):
        """
        https://wiki.fitbit.com/display/API/Fitbit+Subscriptions+API
        """
        url = "{0}/{1}/user/-{collection}/apiSubscriptions.json".format(
            *self._get_common_args(),
            collection='/{0}'.format(collection) if collection else ''
        )
        return self.make_request(url)
```

B.3 exceptions.py

```
import json
```

```
class BadResponse(Exception):
    """
    Currently used if the response can't be json encoded, despite a
    .json extension
    """
    pass


class DeleteError(Exception):
    """
    Used when a delete request did not return a 204
    """
    pass


class HTTPException(Exception):
    def __init__(self, response, *args, **kwargs):
        try:
            errors = json.loads(response.content.decode('utf8'))['errors']
            message = '\n'.join([error['message'] for error in errors])
        except Exception:
            if hasattr(response, 'status_code') and response.status_code == 401:
                message = response.content.decode('utf8')
            else:
                message = response
        super(HTTPException, self).__init__(message, *args, **kwargs)


class HTTPBadRequest(HTTPException):
    """Generic >= 400 error
    """
    pass


class HTTPUnauthorized(HTTPException):
    """401
    """
    pass


class HTTPForbidden(HTTPException):
```

```
    """403
    """
    pass


class HTTPNotFound(HTTPException):
    """404
    """
    pass


class HTTPConflict(HTTPException):
    """409 - returned when creating conflicting resources
    """
    pass


class HTTPTooManyRequests(HTTPException):
    """429 - returned when exceeding rate limits
    """
    pass


class HTTPServerError(HTTPException):
    """Generic >= 500 error
    """
    pass
```

B.4 utils.py

```
def curry(_curried_func, *args, **kwargs):
    def _curried(*moreargs, **morekwargs):
        return _curried_func(*(args+moreargs), **dict(kwargs, **morekwargs))
    return _curried
```