

電気情報工学セミナーⅡ

実験のためのPython & Git入門（機械学習による関数の近似実験）

～ 第1回 イントロダクション・環境構築 ～

池原研究室

イントロダクション

池原研究室の研究対象：マルチメディア信号処理

✓ 実態としてはデジタル画像・映像処理

人が見るための画像処理

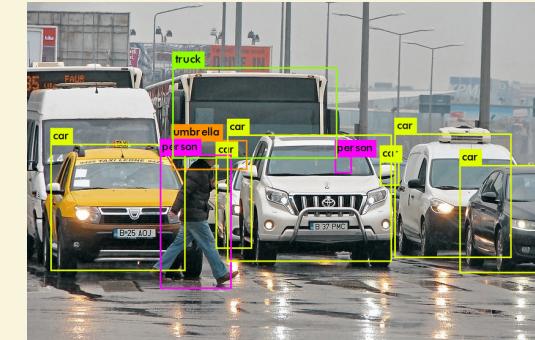
- 画像中のもやを取り除く
- 画像の解像度を上げる
- 映像のフレームレートを増やす
- 暗い画像を明るく鮮やかにする



逆光除去

人の代わりに見るための画像処理

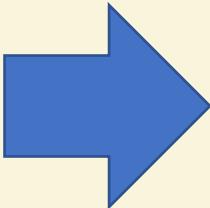
- 自動運転
- 顔認証
- 監視・異常検知
- 医療用画像診断



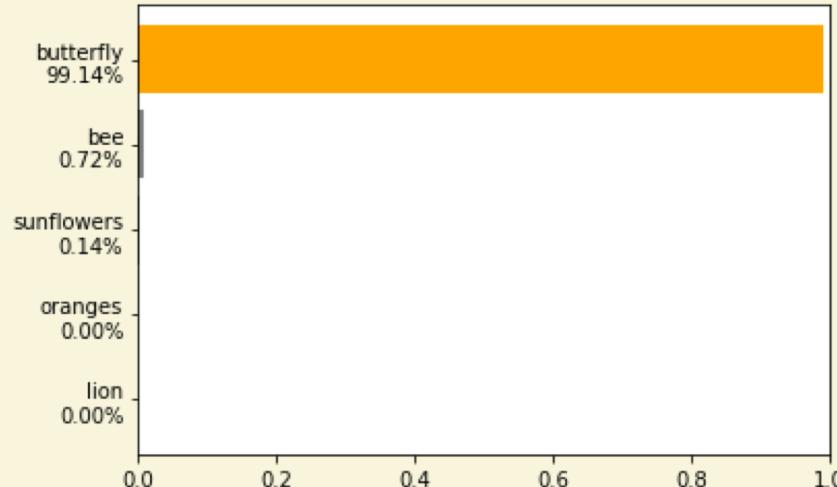
物体認識

画像分類 (Image Classification)

- ✓ 与えられた画像が既定の候補のうちどれに属するかを推測する



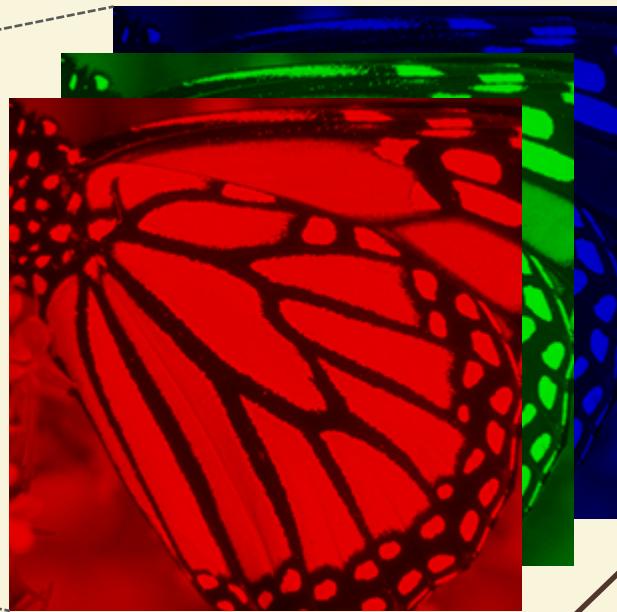
分類候補
リンゴ / バナナ / 蝶 /
...
/ 車 / 傘 / 水筒 / 庭



「人の代わりに見るための技術」の根幹

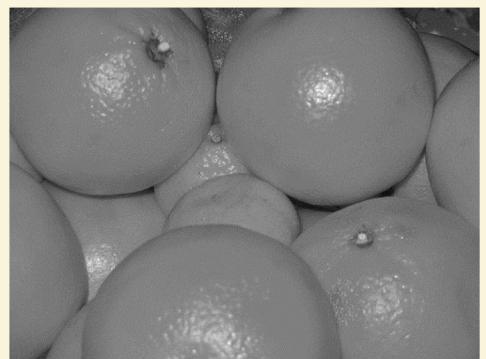
デジタル画像

- ✓ 0～255の数値が縦・横・チャンネル方向に並んだもの
- ✓ 数値は各位置座標における赤(R)緑(G)青(B)の光の強さを表す



```
[[22 26 42 ... 52 49 49]
 [ 12 12 10 ... 51 52 51]
 [[ 30 43 66 ... 59 56 56]
 [ 23 29 38 ... 56 57 56]
 [[ 42 58 85 ... 77 74 74] 54]
 [ 41 47 54 ... 74 75 74] 73]
 [ 80 52 42 ... 74 75 70] 101] 75]
 ...
 [ 98 90 96 ... 151 151 146] 103]]
 [100 97 117 ... 152 150 149]
 [110 109 122 ... 152 150 148]]
```

Q.1：オレンジとバナナの見分け方は？



ルールベースでの画像分類は難しい

Q.2：？に入る数は？

$2 \mapsto 6$

$3 \mapsto 10$

$5 \mapsto 21$

$8 \mapsto 45$

$4 \mapsto ?$

$6 \mapsto ?$

解き方を分解して考えてみる

2 \mapsto 6
3 \mapsto 10
5 \mapsto 21
8 \mapsto 45

4 \mapsto ?
6 \mapsto ?

①共通するルールを見つける

$x \mapsto f(x)$ とする。

f が2次関数つまり

$$f(x) = a_2x^2 + a_1x + a_0$$

と考え

$$a_2 = \frac{1}{2}, a_1 = \frac{3}{2}, a_0 = 1$$

とするとすべての例に当てはまる。

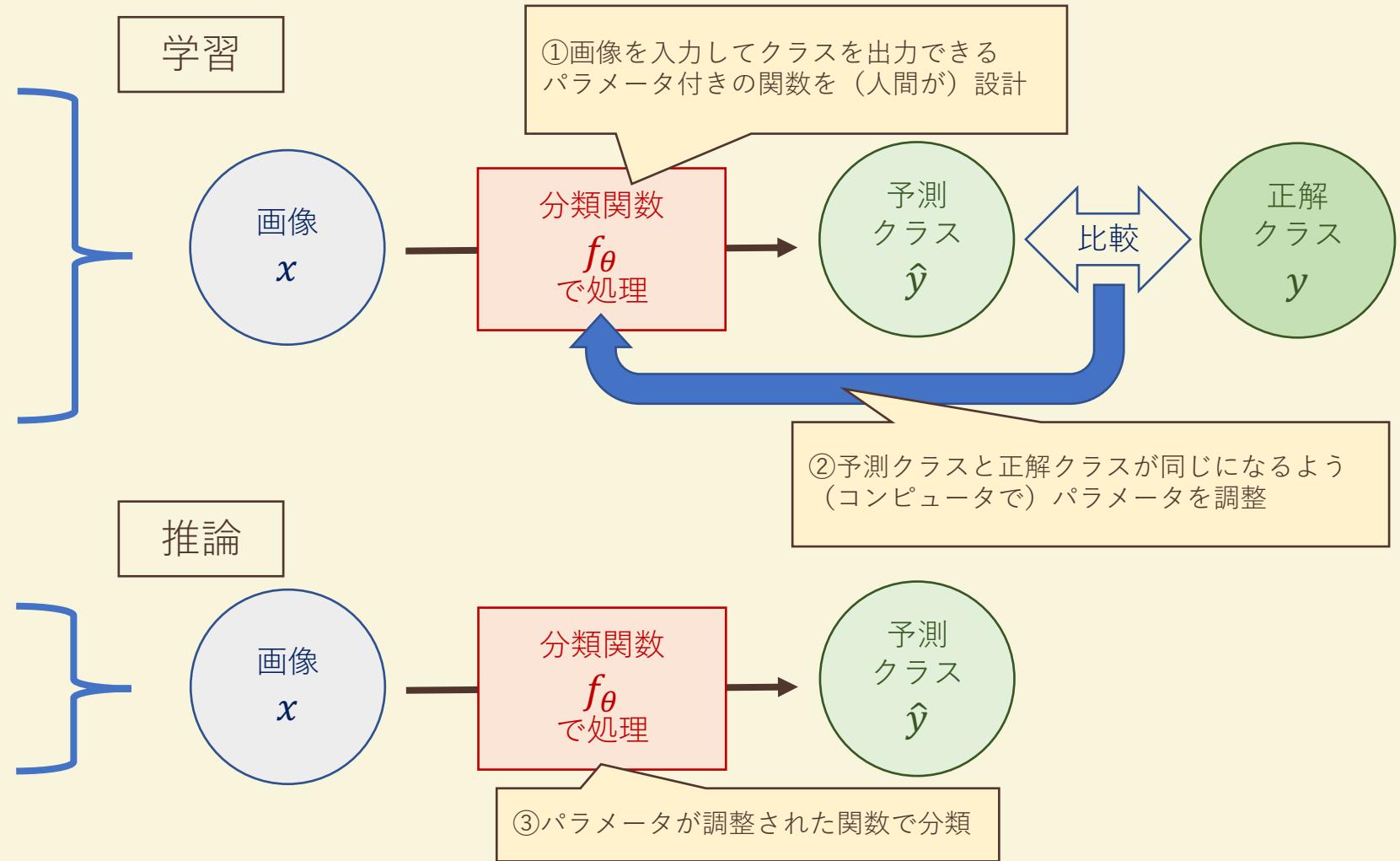
パラメータ付きで定式化

パラメータ推定

$$\begin{bmatrix} 4 & 2 & 1 \\ 9 & 3 & 1 \\ 25 & 5 & 1 \\ 64 & 8 & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 6 \\ 10 \\ 21 \\ 45 \end{bmatrix}$$

機械的に求められる

画像分類を同じように解いてみる



数値の変化の予測と画像処理の関係まとめ

2 \mapsto 6
3 \mapsto 10
5 \mapsto 21
8 \mapsto 45

4 \mapsto ?
6 \mapsto ?

→ オレンジ
→ オレンジ
→ バナナ
→ バナナ

→ ?
→ ?

本講座の
実験テーマ

池原研究室で
扱うテーマ例

学習

入出力の関係を表現する関数を見つける

Step 1

パラメータ付きの関数 f_θ の設計



Step 2

パラメータ θ を推定



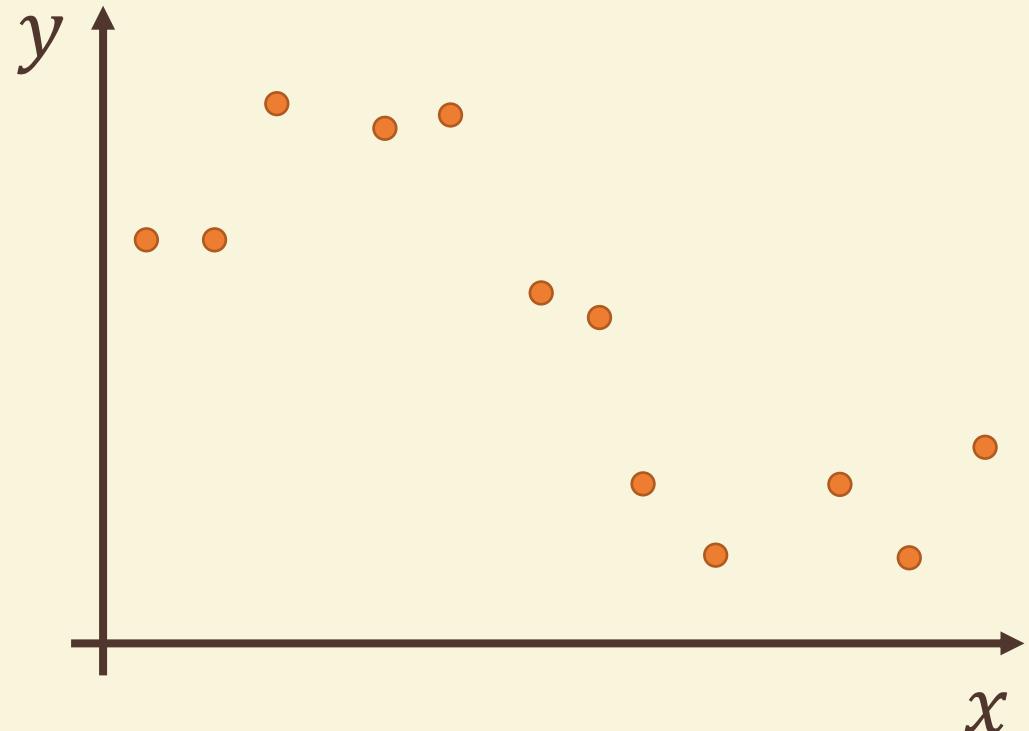
推論

見つけた関数を適用する

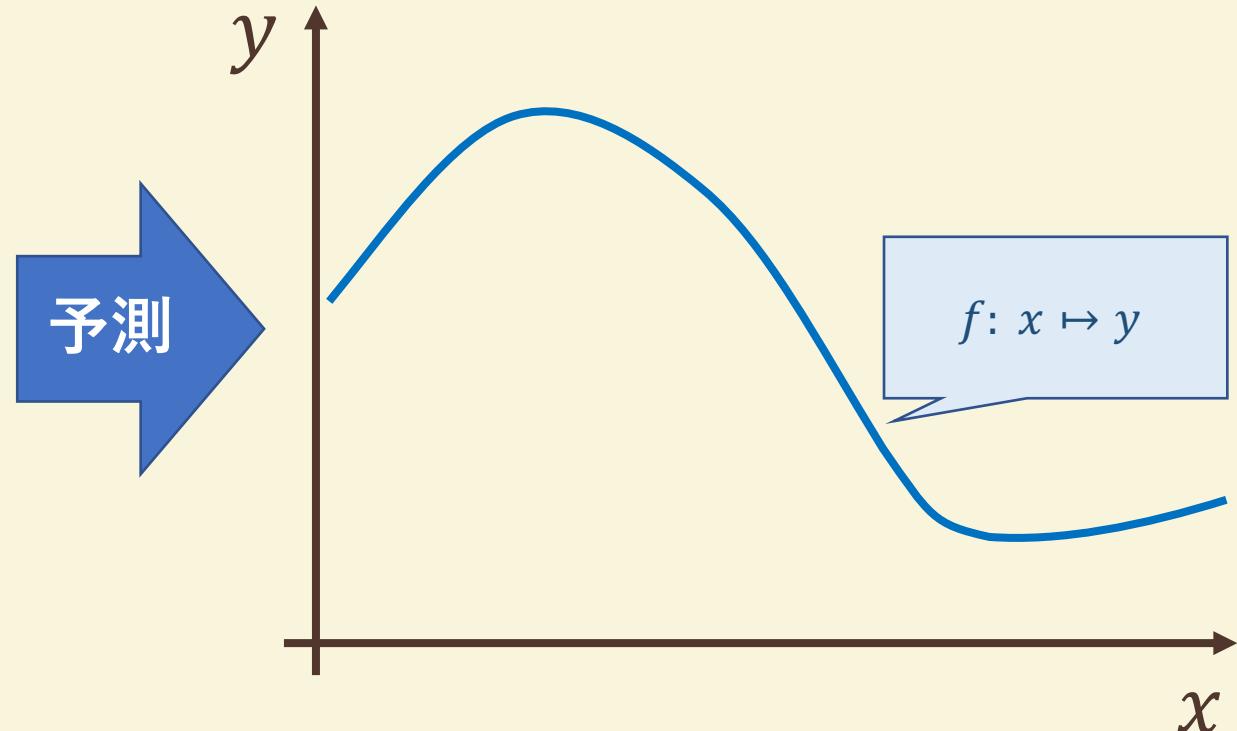
$$\hat{y} = f_\theta(x) \simeq y$$



実験テーマ：回帰分析

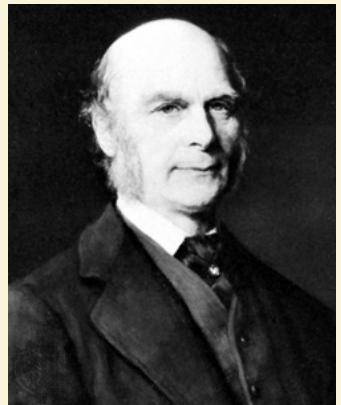


学習データ： $\{(x_i, y_i)\}_{i=1 \dots N}$



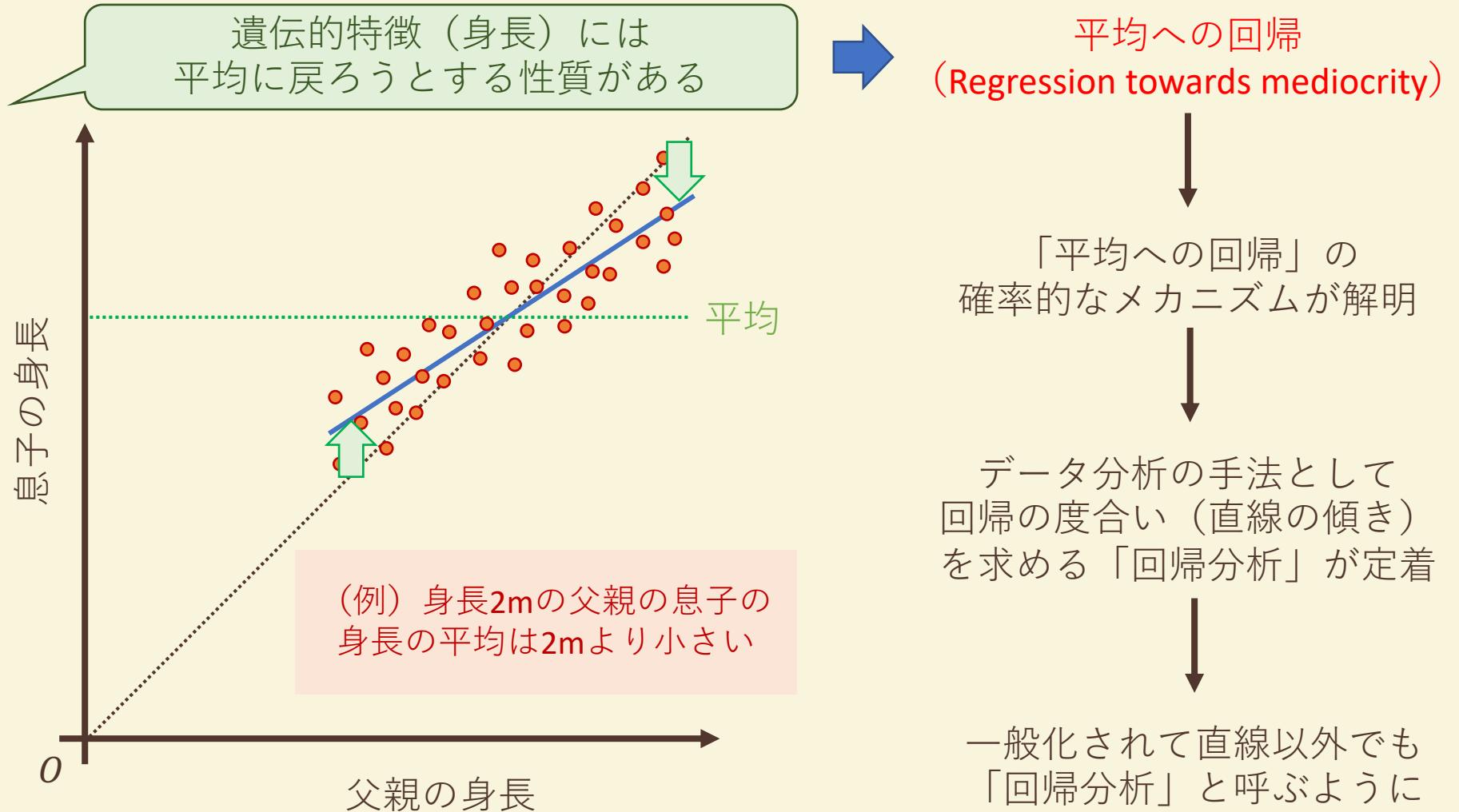
データを生成した関数： $y = f(x)$

雑談：なぜ関数の推定を「回帰」分析と呼ぶのか



Francis Galton
(Darwinのいとこ)

※右図中のデータは
実測値ではなく
説明用のイメージ



Galton, Francis. "Regression towards mediocrity in hereditary stature." *The Journal of the Anthropological Institute of Great Britain and Ireland* 15 (1886): 246-263.

講座概要

①環境構築	②Python入門	③多項式フィッティング
本講座の実験と池原研究室の関連性を学ぶ。また、第2回以降必要な環境を構築する。	Pythonの基本的な文法を書きながら学ぶ。また、コードを書くにあたってGitで履歴を残す方法を学ぶ。	GitHubを活用しつつ、多項式による回帰を実装する。Pythonを用いた数値計算の方法やグラフによる可視化の方法を学ぶ。
④ガウス過程回帰	⑤ニューラルネットワーク	⑥プレゼン発表
ガウス過程回帰を実装する。第3回で実装した多項式フィッティングのコードを活かしつつ、比較するための方法を学ぶ。	ニューラルネットワークによる回帰を実装する。これまでの経験を活かして各自で実装を試みる。	課題として各自が独自の実験を実施し簡単な発表用スライドを作成しておく。各自発表をおこなう。つまり、小規模な研究を体験。

隔週木曜日1・2限実施

休みの週は前日までの申告制で木曜日1・2限にサポート (fukuzaki@tkhm.elec.keio.ac.jp)

※申告しなくても25-421にいる確率は高いが保証はできないため無駄足になる可能性がある。いれば訪ねてきてもかまわない。

講座の目標

学術面

線形代数の応用例を通して
線形性に関するイメージを掴む

実験条件の精査・実験結果の考察を通して
疑問点を見つけ、解決する力を養う

機械学習にできること・できないことや
適用する際の困難を知る

技術面

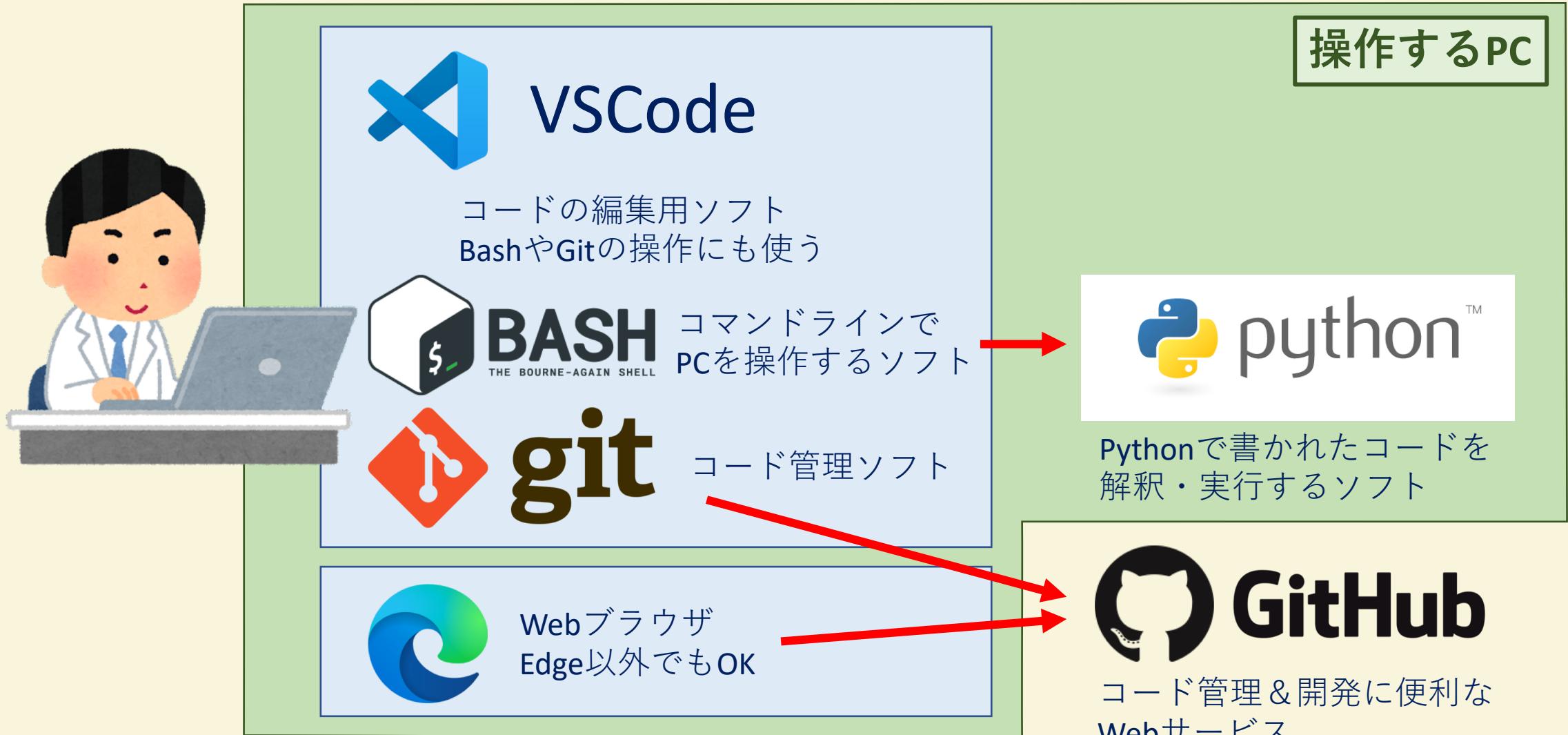
ソフトやプログラムのメッセージを読み
論理的に問題解決できるようになる

自分のやりたいことを整理し
順序立ててプログラムを書けるようになる

(複数人ではなく) 個人の開発で
GitやGitHubを活用する方法の一例を学ぶ

環境構築

構築する環境



Pythonとは

第2回以降
学んでいく



python™

プログラミング言語

「こういう風に書けばこうなる」
という（理想的な）仕様

仕様は年々変化し
バージョンづけられている

今から
インストール

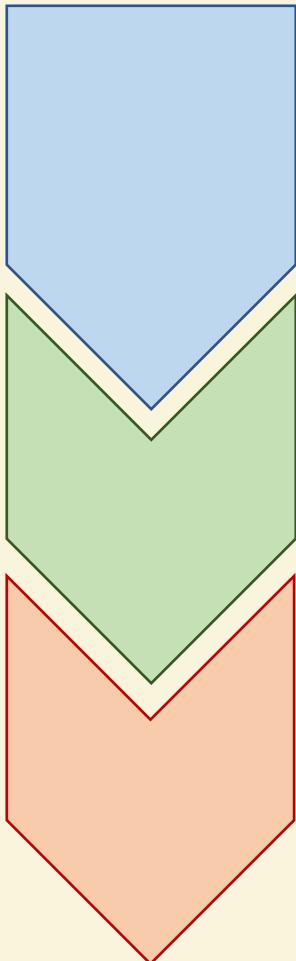
コードを解釈・実行するソフト

言語仕様に沿った挙動を実現

言語仕様のバージョンに対応した
バージョンのソフトが必要

Python 3.10（言語）で書かれたコードをPython 3.10（ソフト）を使って実行する

Python（ソフト）のインストール



- ① 公式サイト (<https://www.python.org/downloads/>) から Python 3.10.8 のインストーラをダウンロード
- ② ダウンロードしたインストーラを起動
- ③ インストーラの指示に従ってインストール

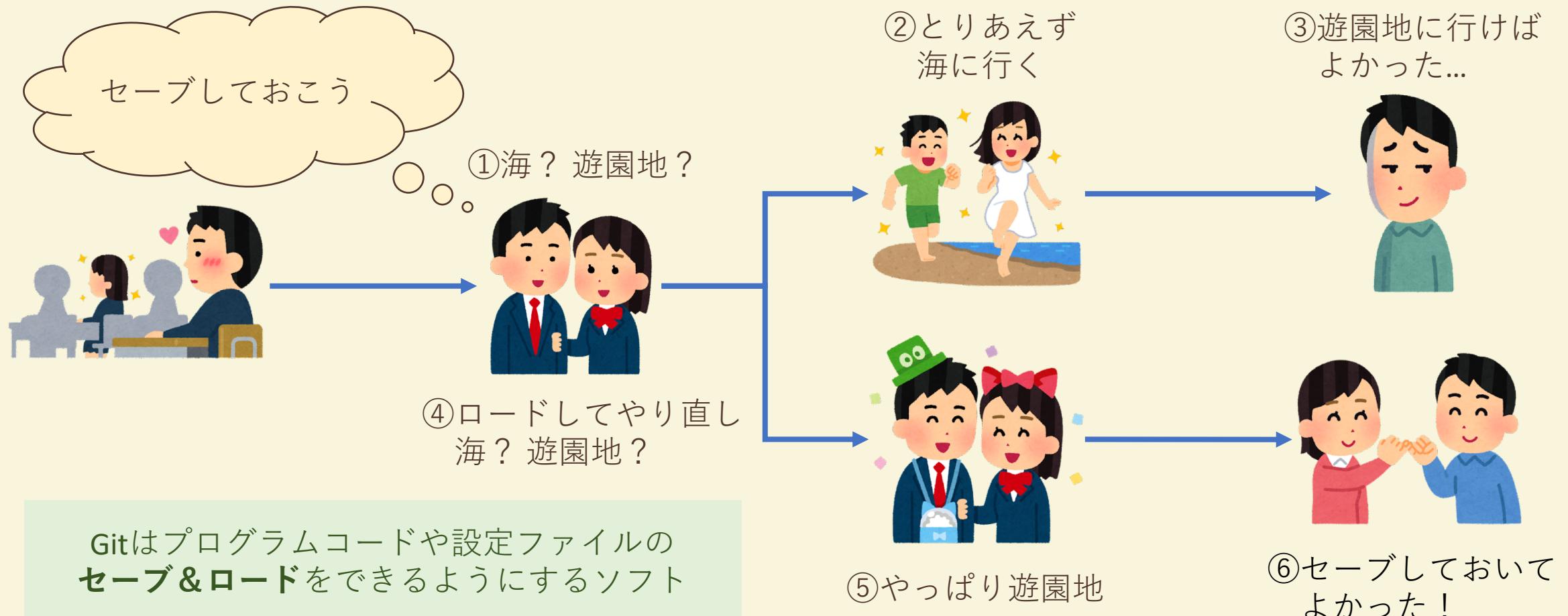
Bashとは



- ✓ 操作内容を文字列で表現可能
- ✓ OSの違いを意識しなくてよい

操作方法の指示が
統一的で簡潔に
「これ打って」でOKに

Gitとは



- ✓ コードの書き換えに失敗しても元に戻せる
- ✓ コードを書き換える前のシミュレーションを再現できる

→ 気兼ねなくコード編集が可能

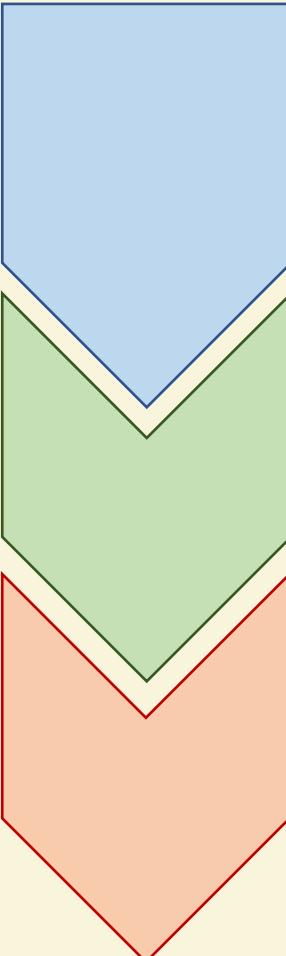
Git & Bashのインストール

Windows

<https://gitforwindows.org/>にアクセスして
インストーラをダウンロード

ダウンロードしたインストーラを起動

インストーラの指示に従ってインストール



MacOS

Terminal.appを開いてGitが
インストールされているか確認

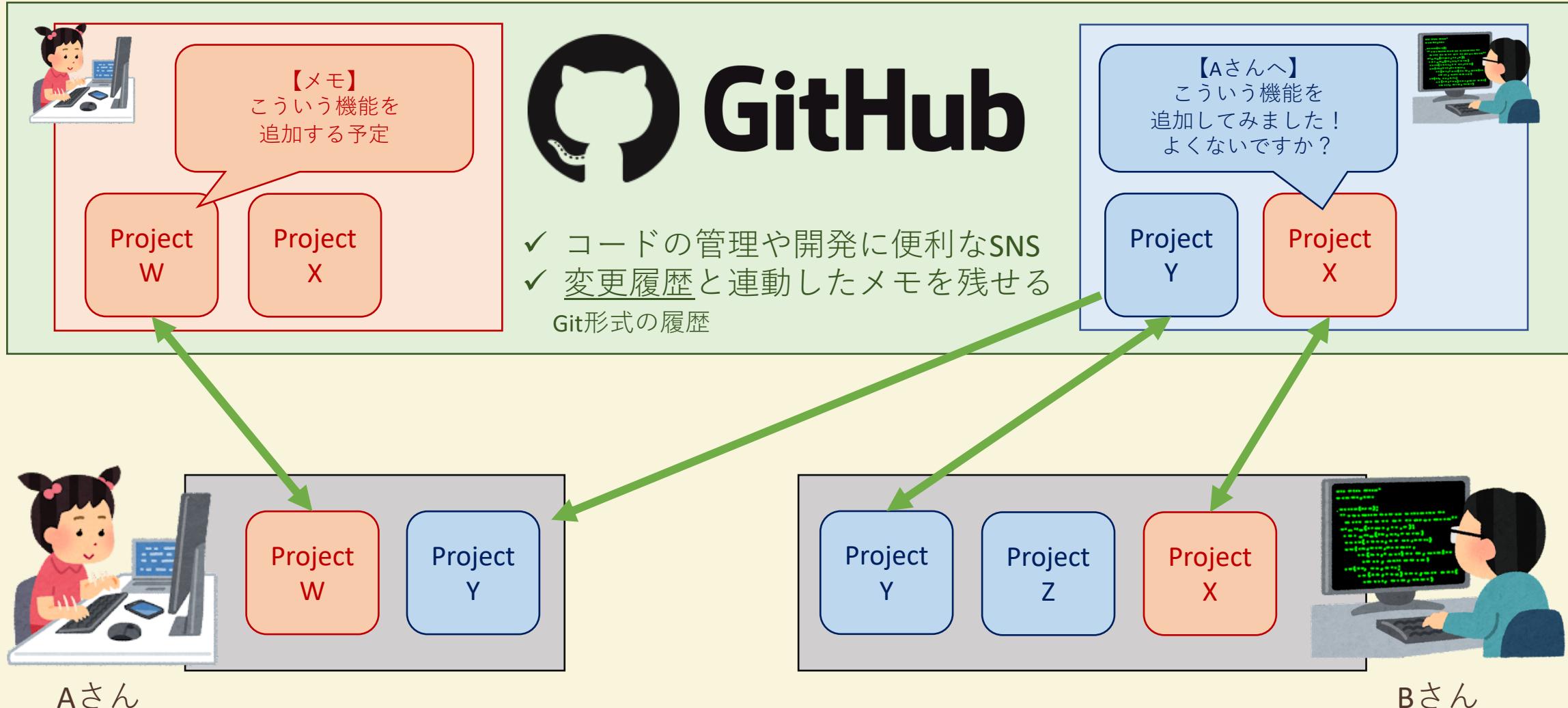
\$ git --version ※コマンド前の \$ は無視

(バージョンが表示されなければ)
https://brew.sh/index_ja の指示に従って
Homebrewをインストール

Gitをインストール

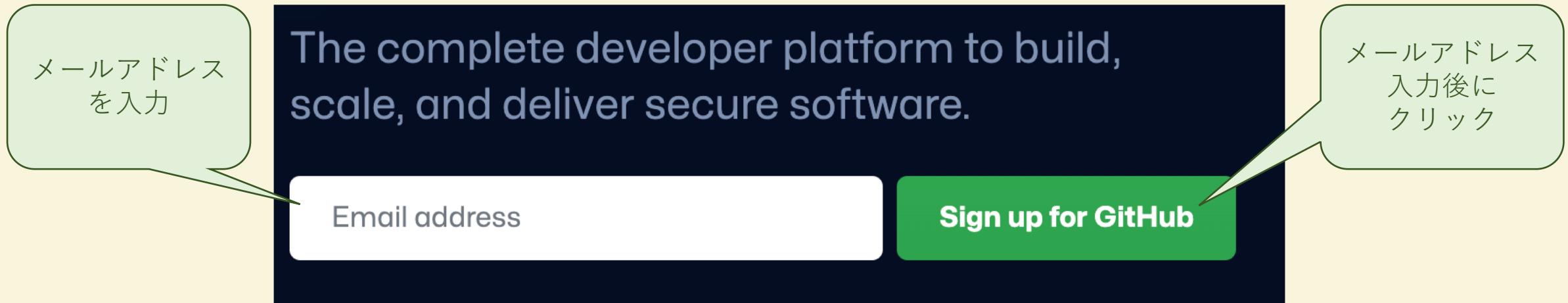
\$ brew install git

GitHubとは



GitHubに登録

<https://github.com/> にアクセス



あとはGitHub側の指示通りに

VSCode (Visual Studio Code) とは



- ✓ コードを書くのに便利なテキストエディタ
- ✓ GitやBashも操作可能



GitHubの操作以外は1画面で開発可能

ファイル一覧を
クリックするだけで
開くことができる

Gitの操作が可能

拡張機能や設定により
自分好みにカスタマイズ可能
(VimやEmacsのキーバインドも◎)

```
classification.py
...
def __init__(self, input_size, hidden_size, output_size):
    super().__init__()
    self.linear1 = nn.Linear(input_size, hidden_size)
    self.linear2 = nn.Linear(hidden_size, output_size)
    self.relu = nn.ReLU()

def forward(self, x):
    x = self.linear1(x)
    x = self.relu(x)
    x = self.linear2(x)
    return x

model = Net(10, 16, 3)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
trainloader = torch.utils.data.DataLoader(
    datasets.MNIST('mnist', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3089,))
                   ])),
    batch_size=64, shuffle=True)
testloader = torch.utils.data.DataLoader(
    datasets.MNIST('mnist', train=False, download=True,
                  transform=transforms.Compose([
                      transforms.ToTensor(),
                      transforms.Normalize((0.1307,), (0.3089,))
                  ])),
    batch_size=64, shuffle=True)

def train():
    model.train()
    for batch_idx, (data, target) in enumerate(trainloader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print(f'Batch {batch_idx} / {len(trainloader)}')

def test():
    model.eval()
    correct = 0
    with torch.no_grad():
        for data, target in testloader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            _, predicted = output.max(1)
            correct += (predicted == target).sum().item()
    print(f'Accuracy: {100 * correct / len(testloader.dataset)}%')

train()
test()
```

タブや分割機能によって
複数のファイルを
編集する場合も便利

コードを解析して
色分けや補完、
ヒントの表示ができる

Bashの操作が可能

VSCODEのインストール

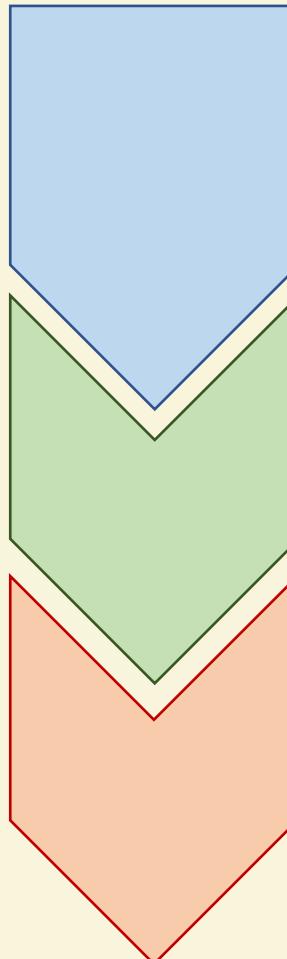
Windows

<https://code.visualstudio.com/download/>

にアクセスして
インストーラをダウンロード

ダウンロードしたインストーラを起動

インストーラの指示に従ってインストール



MacOS

<https://code.visualstudio.com/download/>

にアクセスしてzipファイルをダウンロード

ダウンロードしたzipファイルを解凍

Visual Studio Code.appを
“アプリケーション”フォルダに移動

\$ mv ~/Downloads/Visual Studio Code.app
/Applications

※コマンド前の \$ と改行は無視

VSCodeへの拡張機能の導入

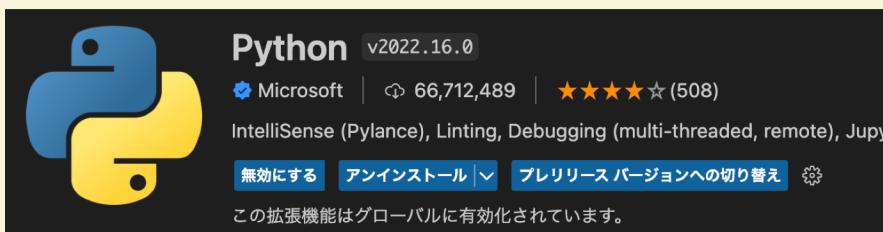


をクリックすると拡張機能の管理メニューが出る

- ✓ 検索窓に以下の拡張機能の名前を入れてインストール

必須

コードを解析して入力補助



必須

自作の関数等への説明文を書くのに便利



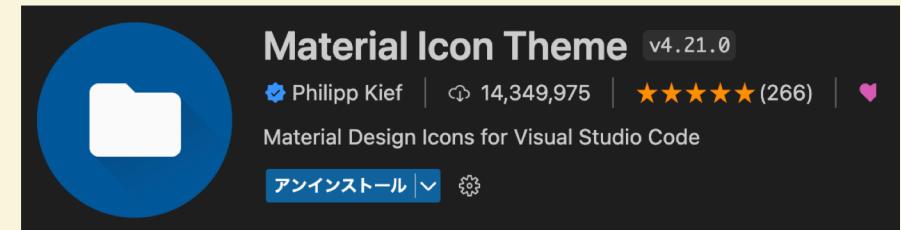
必須

VSCodeの説明を一部日本語化



推奨

ファイルアイコンを表示



VSCODEの設定



> 「設定」で設定用のタブが開ける

「editor.minimap.enabled」で検索 → チェックをはずす

「editor.rulers」で検索 → 「settings.jsonで編集」をクリック

```
{  
    "editor.minimap.enabled": false,  
    "editor.rulers": [80],  
    "git.path": git.exeまでのパス,  
    "terminal.integrated.profiles.windows": {  
        "bash": {  
            "path": bash.exeまでのパス,  
            "args": ["-l"],  
            "icon": "terminal-bash"  
        }  
    },  
    "terminal.integrated.defaultProfile.OS名": "bash"  
}
```

チェックをはずしたことで既に書き込まれている
"＜設定名＞": <設定値> の形で記述されている

Windowsのみ

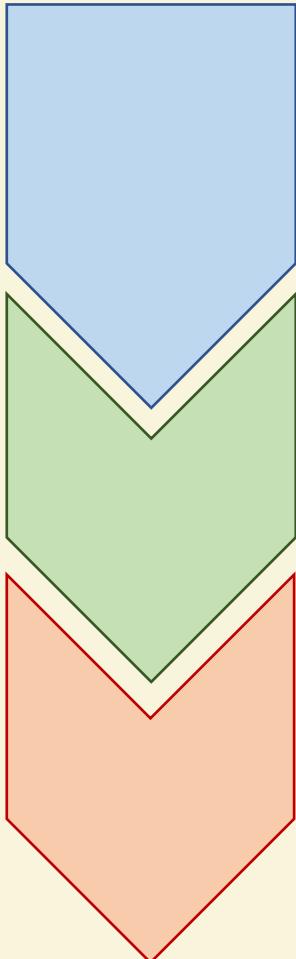
ファイルエクスプローラーの
「ファイルの場所を開く」を
たどってパスを調べる

色付きの部分は調べたパスで
置き換える

※ OS名は windows or osx

codeコマンドを使えるようにする

MacOSのみ



- ① 「Command + Shift + P」でコマンドパレットを起動
- ② 「shell command」と入力
- ③ 「シェル コマンド: PATH内に（以下略）」をクリック

Pythonが使えるかどうかの確認

VSCode内でターミナル（bash）を起動し、以下を実行

```
$ python --version
```

※だいたいのソフトは--versionをつけるとバージョンを表示して終了する

失敗した場合

```
$ code ~/.bash_profile
```

を実行してファイルを開き、以下の1行を追記し保存

```
export PATH=python.3.10.exeまでのパス:$PATH
```

Gitのユーザ登録

「GitHubの」 ではない

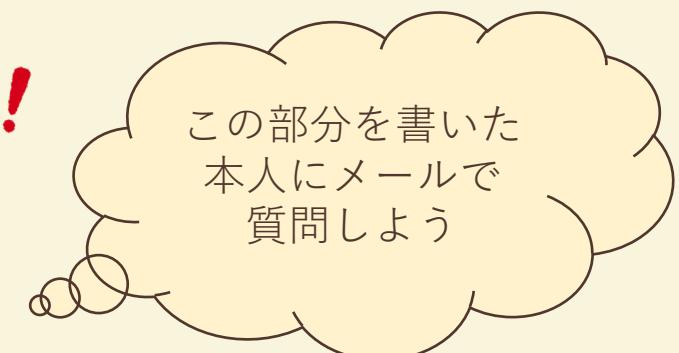
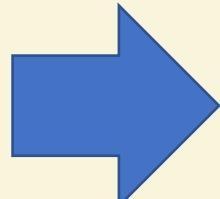
自分の名前

```
$ git config --global user.name "Your Name"  
$ git config --global user.email "your-email@ddress"
```

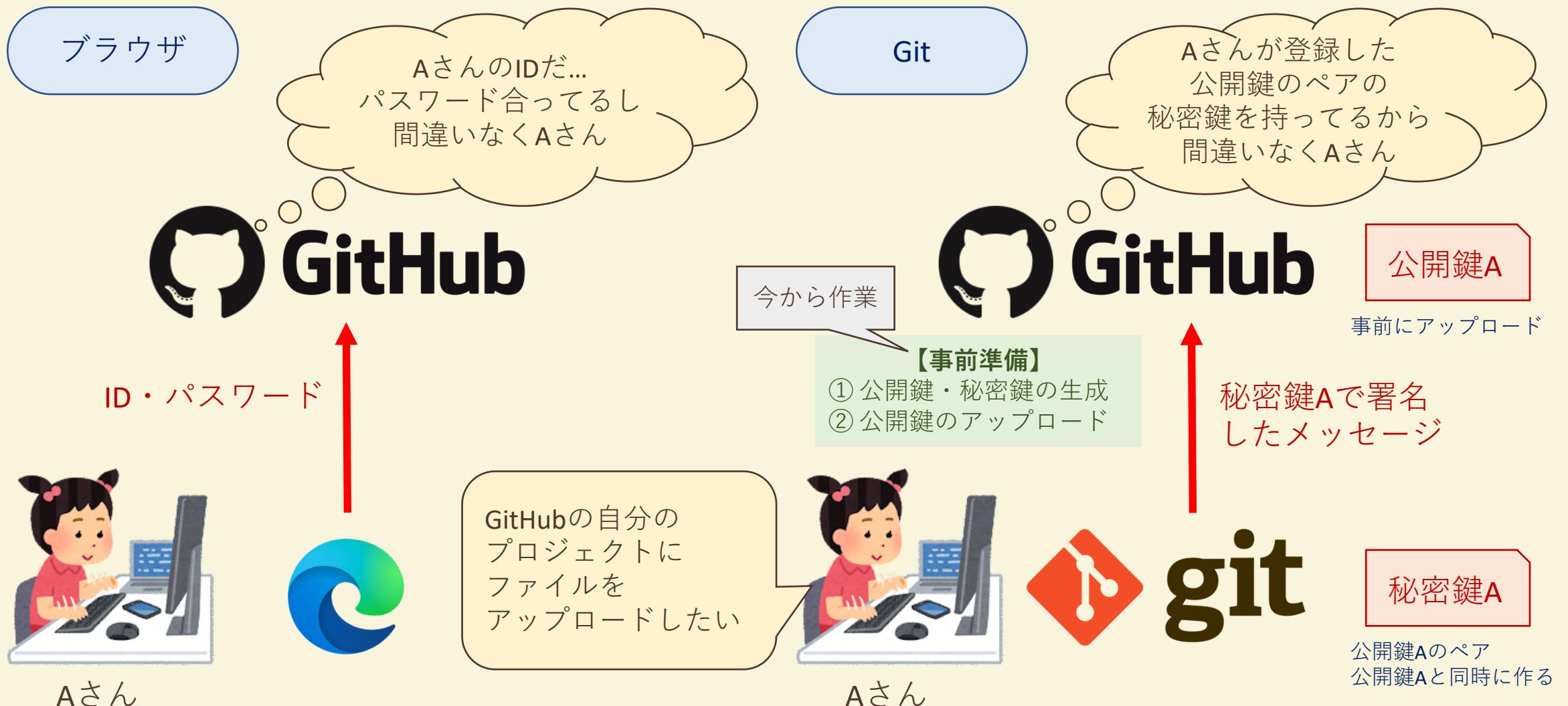
誰が書いたのか
Gitが記録している

自分のメールアドレス

他人が書いた
難しいコード



GitからGitHubに接続するための仕組み



GitHub用の鍵ファイルの生成

```
$ mkdir ~/.ssh  
$ cd ~/.ssh  
$ ssh-keygen -t ed25519 -f github
```

ssh-keygenコマンドは対話式なので説明を読んで回答

生成できたか確認

```
$ ls
```

出力に **github** と **github.pub** が含まれていればOK

秘密鍵

公開鍵 (public key)

【覚えよう：基本コマンド】

mkdir: ディレクトリを作成 (make directory)

cd: ディレクトリの移動 (change directory)

ls: ディレクトリの中身を表示

※ディレクトリ = フォルダ

【覚えよう：～の意味】

～（チルダ）はホームディレクトリ。

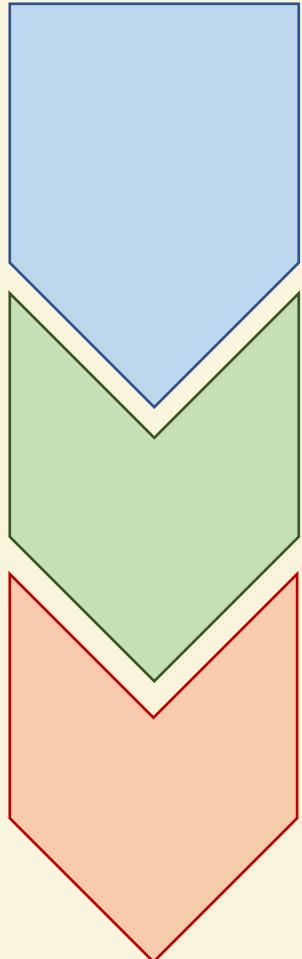
各ユーザが起点として自由に使える。

(例) C:¥Users¥ikehara

bash内では/c/users/ikeharaで表せる

Next → 公開鍵 (github.pub) をGitHubに登録

GitHubへの公開鍵の登録



- ① GitHub (<https://github.com>) を開き
右上のアイコン>Settings で設定ページにアクセス
左のタブから「SSH and GPG keys」をクリック
(<https://github.com/settings/keys> に到達)
- ② 「New SSH Key」をクリック
TitleにPC名（任意）を入力し
Keyに`~/.ssh/github.pub`の中身をコピー＆ペーストする
- ③ 「Add SSH Key」をクリック

```
$ cat ~/.ssh/github.pub
```

ファイルの中身を表示

SSHの設定ファイルを書く

```
$ code ~/.ssh/config
```

を実行してファイルを開き、以下のように内容を編集する

```
Host github github.com
HostName github.com
IdentityFile ~/.ssh/github
User git
```

インデント（字下げ）は
半角スペース2つ

決まった場所・名前のファイルに
決まった形式で設定を書く

```
$ ssh github
```

を実行して先ほどと同じ結果が得られるかを確認する

```
$ ssh -i ~/.ssh/github git@github.com
```

bashの環境変数を設定する

\$ code ~/.bash_profile を実行してファイルを開き、以下の内容を追記する

```
export SEMINAR2=~/dev/seminar2  
export VENV=~/venv
```

\$ exit を実行して一度bashを閉じ、再びbash開く

```
$ echo $SEMINAR2
```

を実行すると設定内容が反映されているとわかる

講義用資料をGitで手に入れる

export した環境変数

省略したら
seminar2 という名前に

```
$ mkdir -p $SEMINAR2  
$ cd $SEMINAR2  
$ git clone github:/fkzk/seminar2 material
```

先ほど設定した
Host の名前

\$SEMINAR2/material というディレクトリに資料が入っている

おまけ

VSCODEの便利なショートカット



> 「キーボード ショートカット」でショートカット設定用のタブが開ける

"ctrl+;"			
コマンド	キー バインド	いつ	ソース
ターミナル: ターミナルにフォーカス	↑ ;	!terminalFocus	ユーザー
表示: 最初のエディター グループ	↑ ;	terminalFocus	ユーザー

