AIプログラミングの準備

Part 1 1日目 はじめに

将来、多くの仕事が人工知能（AI）に置き換わり、大失業時代がやって来る…。そんな話がまことしやかにささやかれている昨今、皆さんいかがお過ごしですか？ どうも、テクニカルライターの中島です。

2017年はまさにAIブームだったわけですが、そんなことどこ吹く風ってなもんで、ノホホンと暮らしていたとある昼下がり、日経ソフトウエアの編集さんから「人工知能プログラミング入門、ヨロシク!」との連絡が…。

編集さん 言語は「Python」、AIライブラリは「scikit-learn」で、お願いしゃぁ～す。

中 島 scikit-learn ??? あっ、ハイ、わかりました。大丈夫っす。

ということで、このお話は、しがないテクニカルライターが「人工知能プログラミング」と格闘する奮闘記です。

▶人工知能の概要を把握

まずは、ネットで「人工知能プログラミング」を調べてみます。人工知能、すなわちAIとは、「artificial intelligence」の略…。いや、これくらいは昔から知ってます。

人工知能という言葉は、1956年にダートマス会議でジョン・マッカーシーさんが命名したらしく、その後「ファジィ理論」や「ニューラルネットワーク」という技法を用いて人工知能を実現しようとした「第2次人工知能ブーム」が到来…。そういえば、ファジィ理論を応用したエアコンとかありましたね（懐かしい）。

そして、2006年の「ディープラーニング」（深層学習）技術の発表が、現在の人工知能ブームにつながっているそうです。ディープラーニングっていうのは、ニューラルネットワークを使う、より高度な「機械学習」のようですね。

Google、Apple、Microsoftなどの米国の大手IT企業も、人工知能の開発プロジェクトを進めていて、2016年にGoogleの子会社DeepMindが作成した「AlphaGo」が、人間のプロ囲碁棋士に勝利したのは記憶に新しいですね。

その後、Googleは注目を浴びたディープラーニングや機械学習を誰でも利用できるように

とライブラリ「TensorFlow」を公開。最近ではプログラミングができない人向けに「Auto ML」という機械学習のサービスも発表しています。

まぁ、この辺の話は至極有名で結構知ってることも多かったのですが、わからないのはその機械学習ってヤツを、どうやってプログラムで利用するのかっていうこと。つまり、具体的な方法ですね。

さらに調べると、TensorFlow以外にもAIのライブラリが多数公開されていて、その中でもscikit-learnは結構有名なようです。つまり、scikit-learnの使い方がわかると、自分で機械学習のプログラムを作れるようになるってわけですか。なるほど…。

▶機械学習に登場する用語を確認

機械学習のことを調べていると、次の用語が頻繁に登場するのでまとめてみました。

教師あり学習

機械学習の手法の一つで、事前に与えられたデータをガイドにして学習を行う。例えば、画像に「猫」が写っていたとして、プログラムがその画像を「猫」と判断できるようにするためには、最初に「猫」とラベリングされた画像を用意し、「猫の特徴」を学習させる。このようにあらかじめ「猫」という正解ラベルを付与したデータを使って学習させる方法を「教師あり学習」と呼ぶ。

教師なし学習

教師あり学習では、あらかじめ人間が画像に「猫」と正解をラベリングする。対して、何もラベリングされていない画像の中から共通の特徴を見つけ、特徴らしいと判断されたデータに対して「これは何の特徴か」を人間が判断する手法を「教師なし学習」と呼ぶ。

クラスタリング

教師なし学習におけるデータの分類手法に「クラスタリング」がある。クラスタリングには、階層的にデータをグループに分類する「階層型」と、特定のグループ数に分類する「非階層型」があり、階層型のクラスタリングには「ウォード法」、非階層的クラスタリングには「k平均法」などのアルゴリズムがある。

学習モデル

機械学習の目的は、データの特徴から、予測するための「モデル」を作ることであ

る。そのモデルを「学習モデル」と呼ぶ。そして、例えば猫の画像認識ができる学習モデルができたあとに、その学習モデルに学習時の画像とは異なる画像を与えることで、その画像に猫の特徴が何パーセント含まれているのかがわかる。

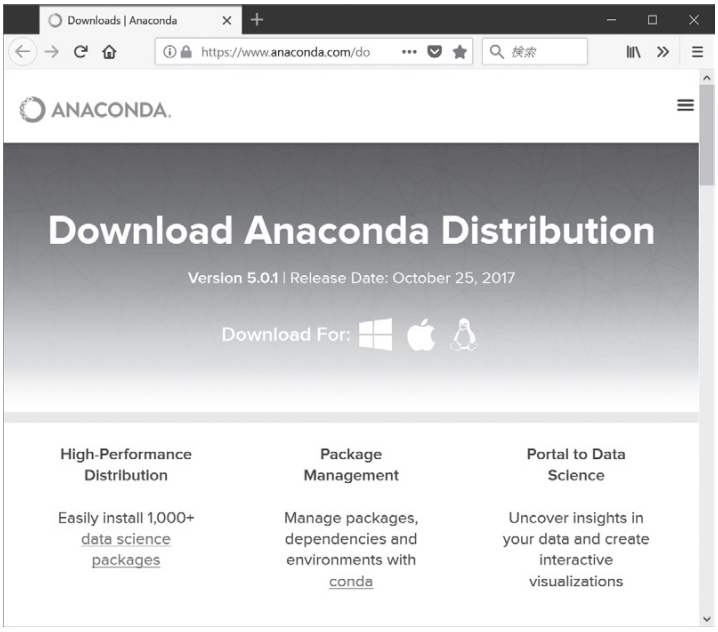
まとめている自分も、まだピンと来ない部分だらけですが、欲張らず機械学習の基本を理解するようがんばりたいと思います。では、学習環境の構築から始めます。

Part 2 Anacondaのインストール

今回、機械学習のプログラミングに利用する言語はPythonです。したがって、Pythonの公式サイトからPythonのプログラミング環境をダウンロードするところから始めるのが良いと思ったのですが、もっと簡単な方法がありました。それは、「Anaconda」（アナコンダ）をインストールすることです。

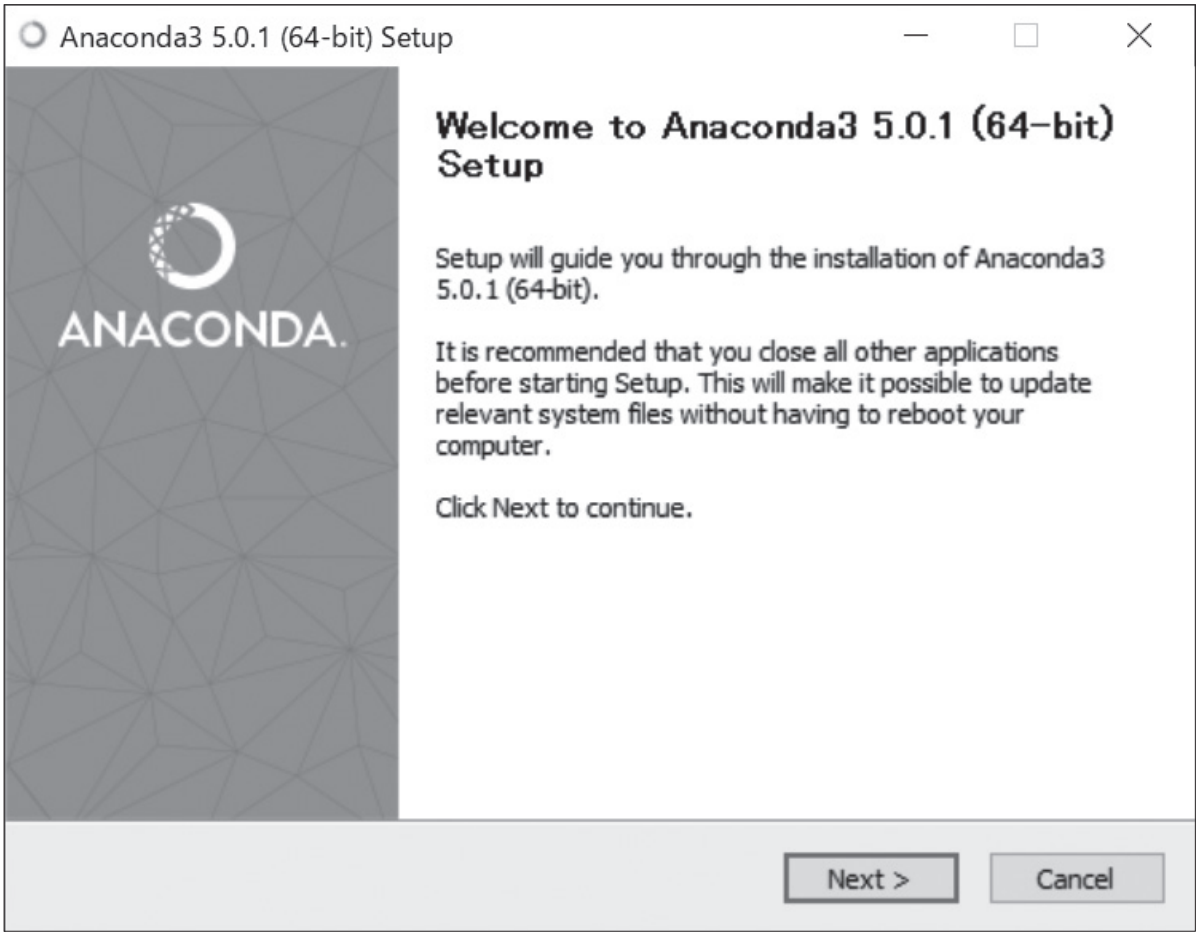
Anacondaは、Continuum Analytics社が提供しているPythonのディストリビューションです。Python本体だけでなく「NumPy」「scikit-learn」「Jupyter Notebook」「Pandas」「matplotlib」といった、機械学習に必要なライブラリやツール類も含まれているという、超便利なディストリビューションになっています。もちろん、無料で使えます。Anacondaは、公式サイトからダウンロードします（**図1**）。

図1●Anacondaのダウンロードサイト（https://www.anaconda.com/download/）



Anacondaにはいろいろなバージョンがありますが、ここでは「Anaconda 5.0.1」のWindows用で「Python 3.6 version」の64ビット版をダウンロードしてインストールしました（**図2**）。ダウンロードしたインストーラのファイル名は「Anaconda3-5.0.1-Windows-x86_64.exe」です。

図2●Anacondaのインストール



ちなみに、Anaconda 5.0.1は2017年の10月にリリースされたもので、原稿執筆時点（2018年1月）での最新版でした。インストール作業はインストーラの指示に従うだけなので簡単です。

インストールが完了すると、「Anaconda3」のフォルダーが［スタート］メニューに登録されているので、試しに「Anaconda Prompt」を起動してみます（**図3**）。

「Anaconda Prompt」のウインドウが開くので、「python」と入力しましょう。するとPythonのインタプリタが起動します。次のコードを入力して動作確認を行いましょう。

```
>>> print("Hello Python!")
Hello Python!
```

「>>>」の後ろに「print("Hello Python!")」と入力して、「Hello Python!」と表示されたらOKです（**図4**）。

動作確認が済んだら、ウインドウを閉じておきましょう。

図3●Anaconda Promptの起動

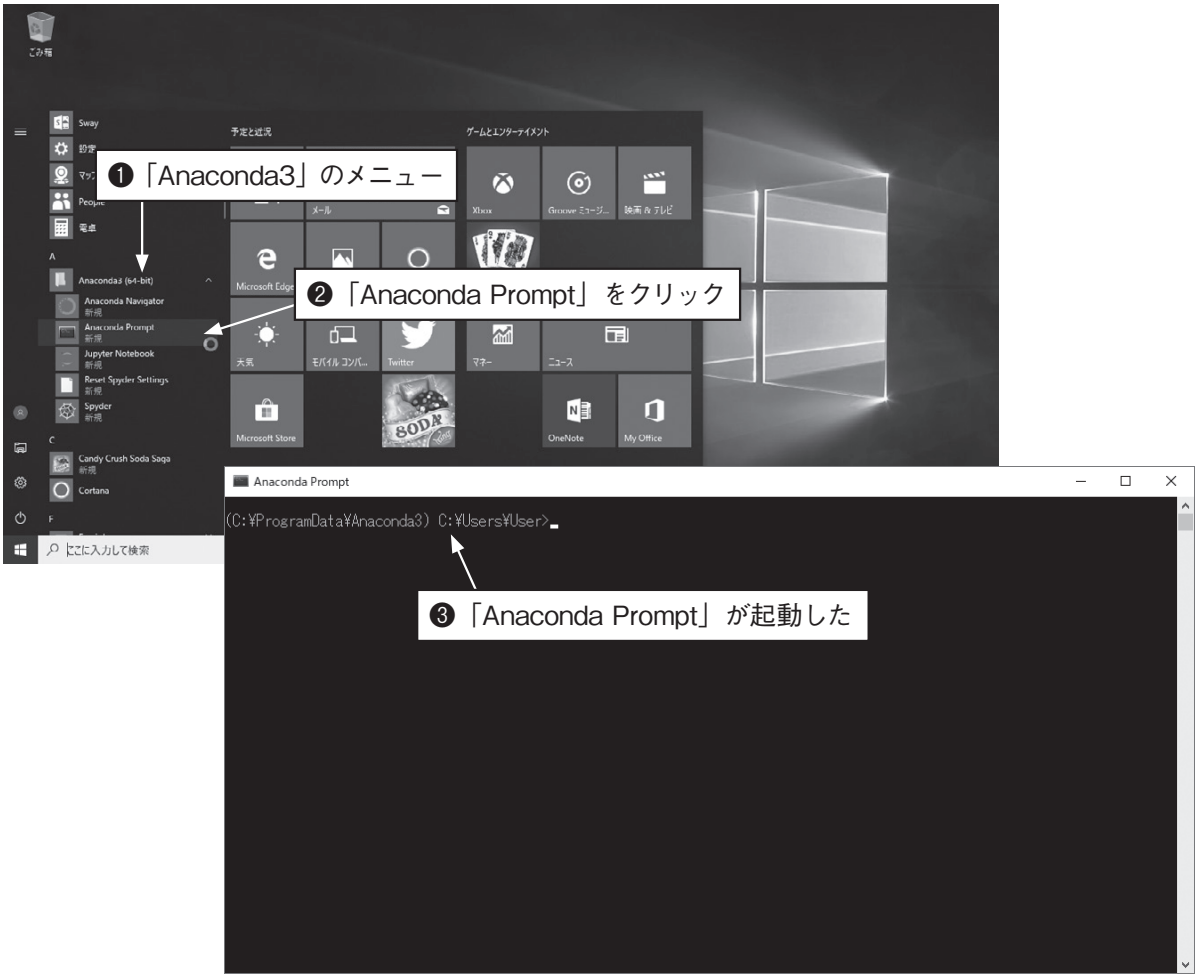
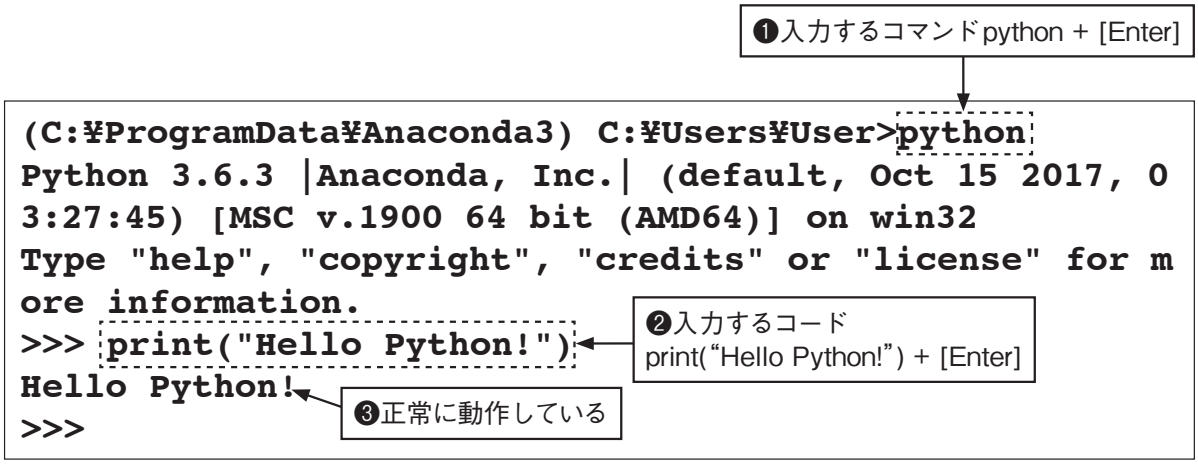


図4●コードを入力・実行して、動作確認を行う

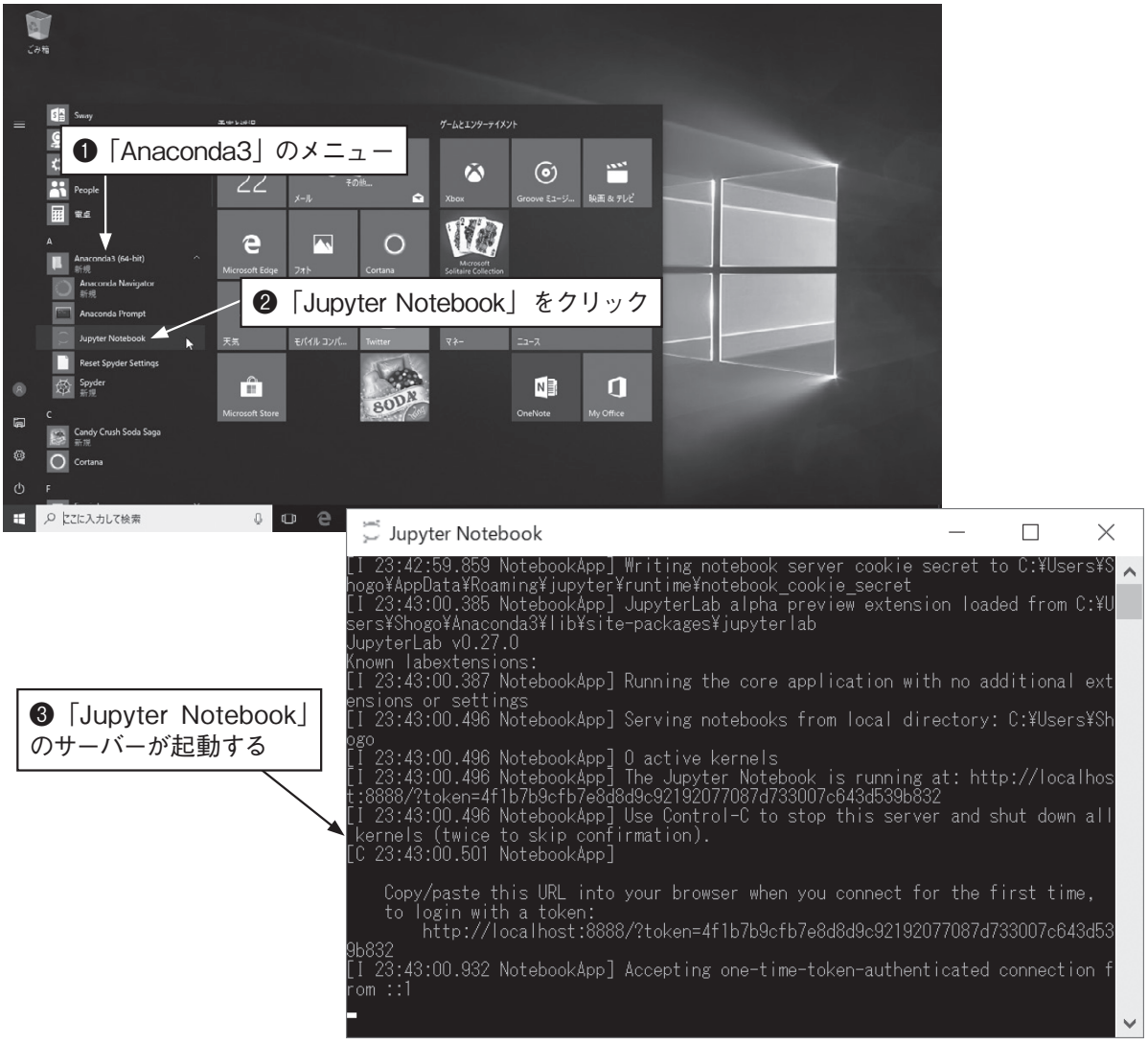


1日目 Part 3 Jupyter Notebook

機械学習の解説では、必ずと言ってよいほどデータや予測結果をグラフで表示します。都合が良いことに、Anacondaには「Jupyter Notebook」という、Pythonのコードとグラフを同時に表示できるプログラミングツールが付いています。コードを記述して実行すると、コードのすぐ下にグラフを表示してくれます。ここではこの便利なJupyter Notebookを使いましょう。

Jupyter Notebookは、スタートメニューの「Anaconda3」のフォルダーにある「Jupyter Notebook」を選択することで起動します（**図1**）。

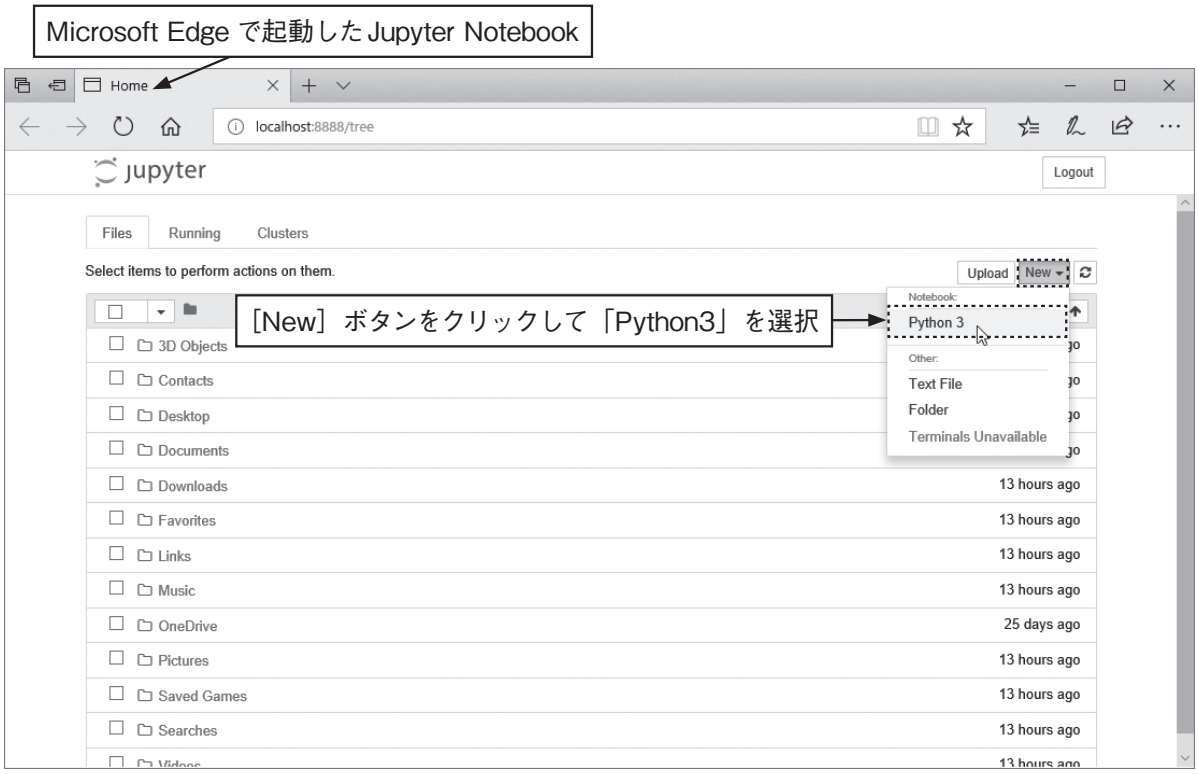
図1●Jupyter Notebookの起動



Jupyter Notebookは、ブラウザで動作するサーバータイプのツールです。ここではブラウザとしてMicrosoft Edgeを使いますが、他のブラウザでも問題はないでしょう。

ブラウザにJupyter Notebookが表示されたら、右上の[New]ボタンをクリックして「Python3」を選択します（**図2**）。

図2●Jupyter NotebookでPython3を選択



Jupyter Notebookでは、「セル」と呼ばれるスペースにPythonのコードを入力します。その後、[run cell, select below]ボタンをクリックしてコードを実行します。先ほどのコードを、セルに入力します。

```
In [1]: print("Hello Python!")
```

実行してみましょう（**図3**）。

ちなみに、[run cell, select below]ボタンにはショートカットキーがあります。セル内のプログラムを実行する際は[Ctrl]+[Enter]キーを押すか、[Shift]+[Enter]キーを押します。[Shift]+[Enter]キーは、実行後カーソルが下のセルに移動します。

また、アイコンには、各種のコマンドが割り当てられています（**図4**）。

Jupyter Notebookには、データの保存や読み込み、エクスポートの機能があります。

Jupyter Notebookでのコードの記述と実行結果をそのまま保存したい場合は「.ipynb」形式でファイルに保存するのが良いでしょう。もちろん、.ipynb形式のファイルを読み

込むことも可能です。

また、HTML形式でエクスポートしたり、Pythonのコードだけを出力することもできます。

図3●Jupyter Notebookで実行

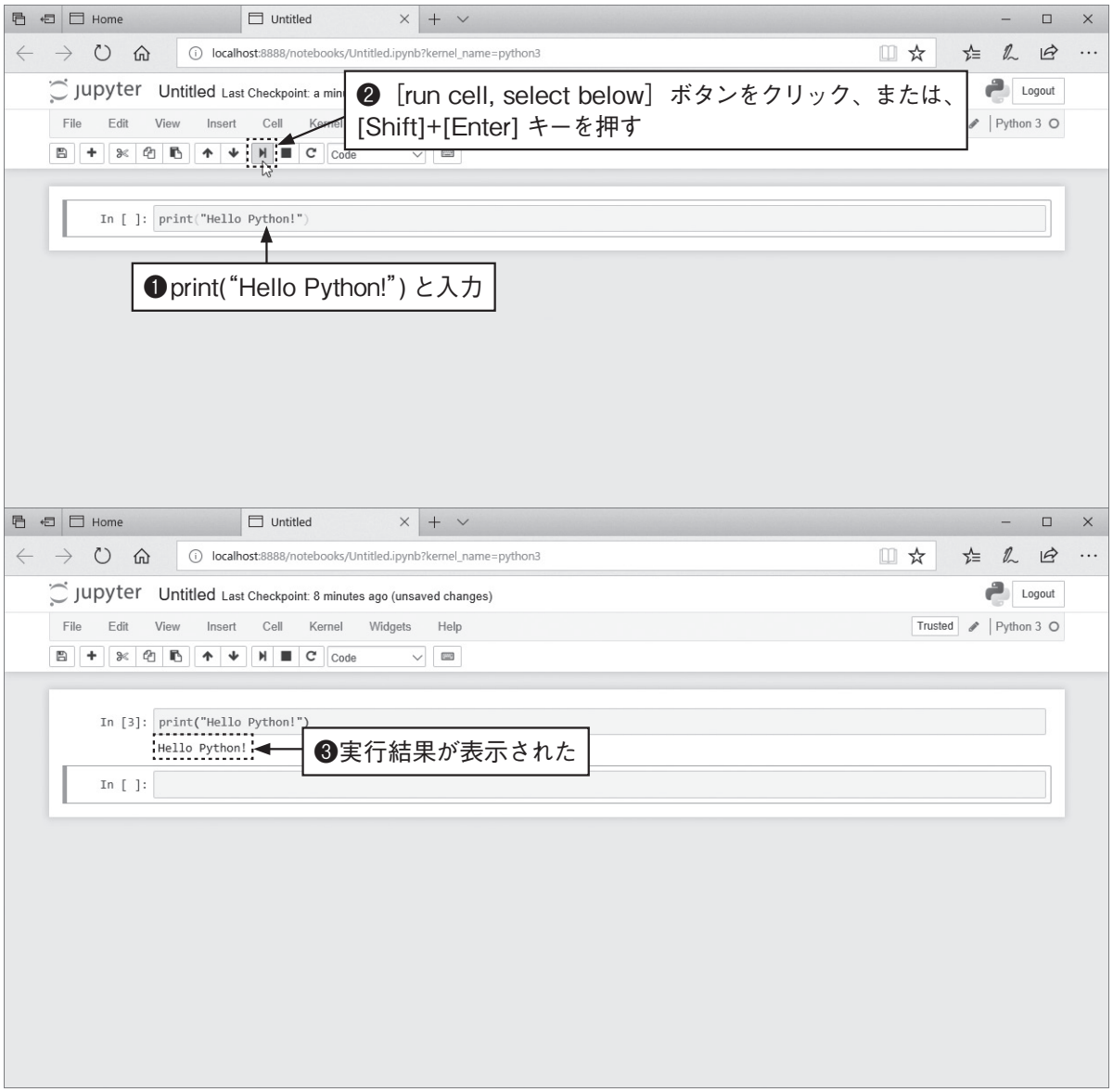
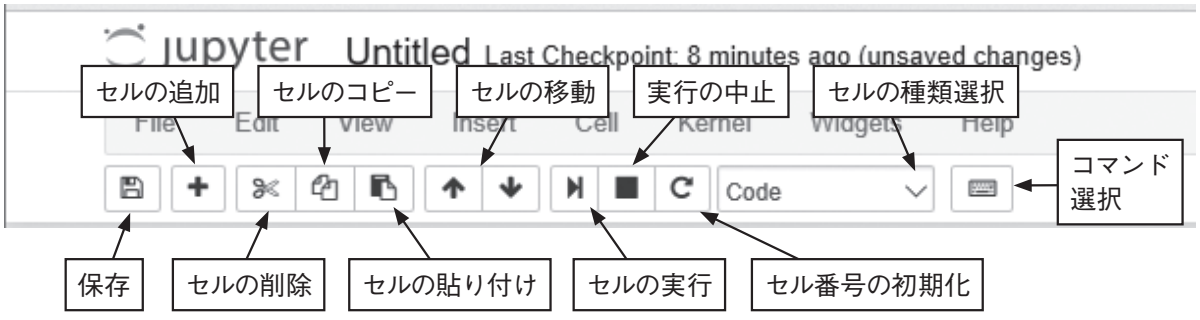


図4●Jupyter Notebookのアイコン



1日目

Part 4 NumPy

機械学習で利用するライブラリの使い方を調べていきます。実行はJupyter Notebookで行いますので、起動しておきましょう。

最初のライブラリは、NumPyです。NumPyは、数値計算を効率的に行うためのモジュールで、今回使うAIライブラリの「scikit-learn」はNumPyを利用しています。

まず、念のためNumPyがインストールされているか確認しましょう。NumPyをnpという名でimportしてみます。

Jupyter Notebookのセルに次のように入力して、[run cell, select below]ボタンをクリックするか[Shift]+[Enter]キーで実行します。

```
In [1]: import numpy as np
```

もし、何らかの理由でNumPyのインストールに失敗している場合はエラーが出ます。その場合は、「Anaconda Prompt」で「pip install numpy」と入力して、再インストールしてください。

▶NumPyで配列を操作

NumPyで配列を作るにはarray関数を使います。例えば、要素が3、5、8の配列は、次のように作ります（**図1**）。

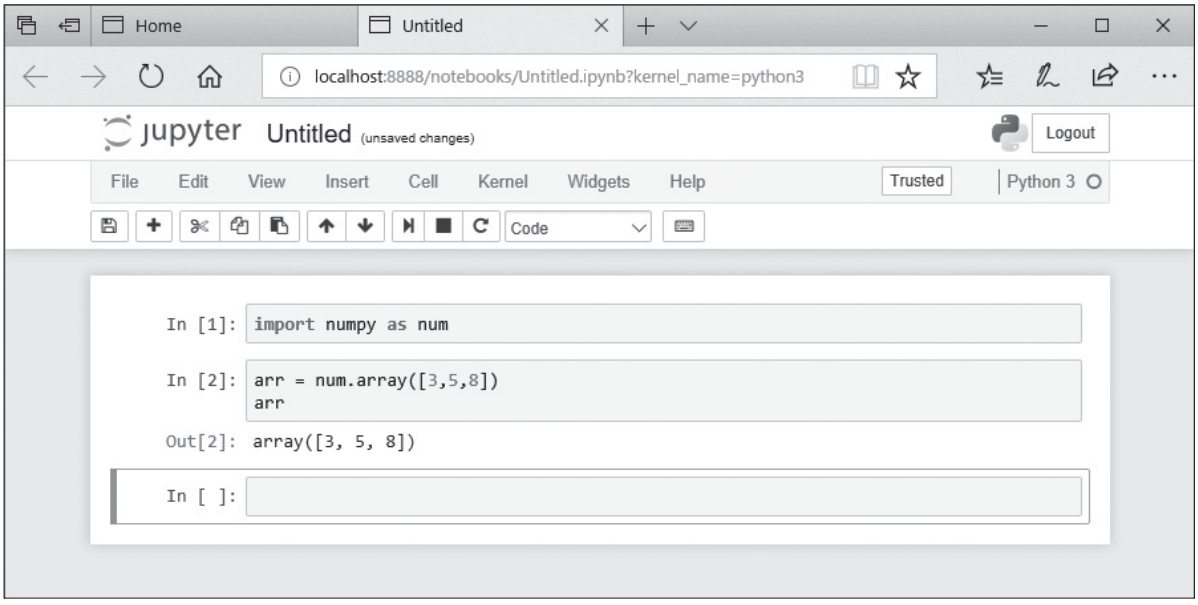
```
In [2]: arr = np.array([3,5,8])
        arr
Out[2]: array([3, 5, 8])
```

2次元配列なら、次のようにします。

```
In [3]: arr = np.array([[3,5,8], [1,4,9]])
        arr
Out[3]: array([[3, 5, 8],
               [1, 4, 9]])
```

NumPyの配列はnumpy.ndarrayクラスのオブジェクトなので、高速に動作する上、便利な関数や属性が多く用意されています。例えば、各次元の要素数が知りたい場合

図1●NumPyのテスト



は、shape属性を参照します。

```
In [4]: arr.shape
Out[4]: (2, 3)
```

次のようにすると、0～1のランダムな値で、要素を初期化することができます。

```
In [5]: arr = np.random.rand(2, 3)
        arr
Out[5]: array([[ 0.5735834 ,  0.92614039,  0.70211732],
               [ 0.34815397,  0.99924502,  0.18050367]])
```

▶ベクトルと行列の演算

配列をベクトルや行列として計算することができます。例えば、次のような配列を用意して5を掛けると、それぞれの要素が5倍になります。

```
In [6]: arr = np.array([1,2,3])
        arr = arr * 5
        arr
Out[6]: array([ 5, 10, 15])
```

同じ形の配列同士の計算は、同じ場所の要素同士が計算されて返ります。

```
In [7]: arr1 = np.array([[1,2,3],[2,3,4]])
        arr2 = np.array([[3,4,5],[4,5,6]])
        arr = arr1 + arr2
        arr
Out[7]: array([[ 4,  6,  8],
               [ 6,  8, 10]])
```

転置を行うには、配列のT属性を参照します。

```
In [8]: arr1 = np.array([[1,2,3],
                         [2,3,4]])
        arr2 = arr1.T
        arr2
Out[8]: array([[1, 2],
               [2, 3],
               [3, 4]])
```

dot関数を使うと、ベクトルの内積や行列の積を求めることができます。書式は、**図2**のようになります。

図2●dot関数の書式

```
numpy.dot(a, b, out = None)
```

引数	説明
a	左からかけるベクトルまたは行列
b	右からかけるベクトルまたは行列
out	結果を格納する代替配列
返り値	ベクトルの内積の結果や、行列の積の結果

ベクトルの内積とは、各要素の積をすべて足し合わせた値です。例えば、次のarr1とarr2のベクトルの内積は、1×2 + 2×3 + 3×4で20になります。

```
In [9]: arr1 = np.array([1,2,3])
        arr2 = np.array([2,3,4])
        np.dot(arr1,arr2)
Out[9]: 20
```

行列の積では、横の並び、縦の並びの組の同じ順番の数同士を掛けたものを足します。例えば、次のarr1とarr2の行列の積は、[1×5 + 2×7, 1×6 + 2×8], [3×5 + 4×7, 3×6 + 4×8]になります。

```
In [10]: arr1 = np.array([[1,2], [3,4]])
         arr2 = np.array([[5, 6], [7,8]])
         np.dot(arr1, arr2)
Out[10]: array([[19, 22],
                [43, 50]])
```

▶NumPyの統計関数

配列の要素の平均は、mean関数を使うと簡単に計算できます（**図3**）。

図3●mean関数の書式

```
numpy.mean(a, axis = None, dtype = None, out = None, keepdims =False)
```

引数	説明
a	平均を求めたい配列
axis	どの軸(axis) に沿って平均を求めるか
dtype	計算結果を格納するための配列
out	結果を格納する代替配列
keepdims	返す配列の軸(axis) の数をそのままにする
返り値	指定した配列の要素の平均、もしくは平均を要素とする配列

例えば、0 ～ 9までのランダムな整数の配列を生成して、その平均を計算してみましょう。ランダムな値の配列は、random.randint関数を使って作ります。第1引数は下限、第2引数は上限（この数は含まない）、第3引数は要素数です。

```
In [11]: r = np.random.randint(0, 10, 10)
         r
Out[11]: array([9, 1, 1, 5, 8, 0, 4, 8, 7, 6])
```

この配列のすべての要素の平均は、次のようになります。

```
In [12]: m = np.mean(r)
         m
Out[12]: 4.9000000000000004
```

標準偏差は、std関数で求めることができます（**図4**）。

図4●std関数の書式

```
numpy.std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=
<class numpy._globals._NoValue>)
```

引数	説明
a	標準偏差を計算したい配列
axis	どの軸(axis) に沿って平均を求めるか
dtype	計算結果を格納するための配列
out	結果を格納する配列
ddof	データ個数 N ではなく “N - ddof” で割る
keepdims	True にすると出力される配列の次元数が保存される
返り値	指定された範囲での標準偏差を要素とする配列、または値

先ほどの配列rの標準偏差は、次のようになります。

```
In [13]: s = np.std(r)
         s
Out[13]: 3.1128764832546763
```

集計する関数として、sum関数があります。このsum関数にはaxis（軸）を表す引数があり、2次元配列に関してどの向きに集計を取るのか指定できます。

```
In [14]: arr = np.array([[1,2,3],
                         [2,3,4]])
         np.sum(arr, axis=0)
Out[14]: array([3, 5, 7])

In [15]: np.sum(arr, axis=1)
Out[15]: array([6, 9])
```

2点間の距離（ベクトルの長さ）、いわゆる「ユークリッド距離」を求める場合は、linalg.norm関数を使います。座標（x, y）として、a地点(2, 5)からb地点(7, 8)へのユークリッド距離を求めるには、次のように記述します。

```
In [16]: a = np.array([2, 5])
         b = np.array([7, 8])
         np.linalg.norm(b - a)
Out[16]: 5.8309518948453007
```

1日目

Part 5 Pandas

Pandasは、Pythonでデータの分析、解析を支援するライブラリです。機械学習では、データをPandasの「DataFrame」に変換して、操作したり表示したりすることが多いので、ここではDataFrameを少し調べておきます。

DataFrameは、RDB（関係データベース）のテーブルやCSV、Excelのテーブルのような形式のデータ構造のようです。いわゆる「表」のようなイメージでしょうか。

▶DataFrameを作る

では、DataFrameを作ってみます。最初に、NumPyとPandasをインポートします。

```
In [1]: import numpy as np
        import pandas as pd
```

とりあえず、DataFrameのコンストラクタにリストを渡して、DataFrameオブジェクトを作ります。リストは、名前、国籍、年齢、誕生日の順に並んでいますが、表にしたとき見やすいように転置しています。

```
In [2]: df = pd.DataFrame([["Nakajima","Smith","Chen"],
                           ["Japan","USA","China"],
                           [51,25,39],
                           ["11/20","3/5","8/29"]] ).T
```

これで、DataFrameの原型ができました。さらに、表の列名をcolumns関数で付けます。

```
In [3]: df.columns = ["Name","Country","Age","Birthday"]
```

もし、表に行名が必要ならindex属性にリストをセットします。

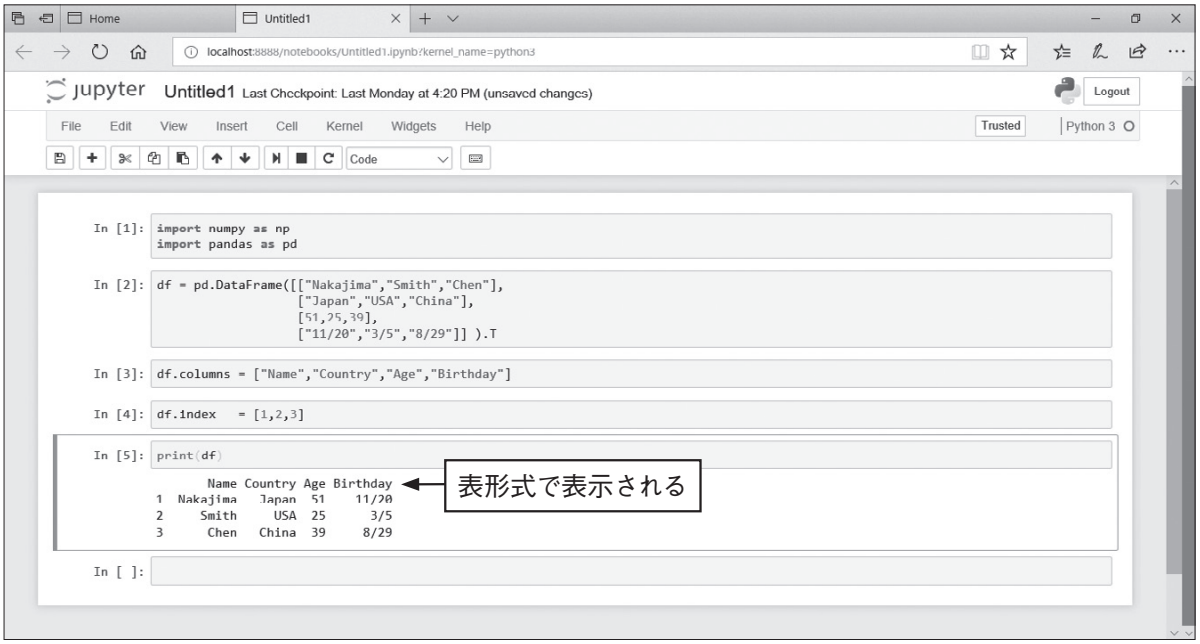
```
In [4]: df.index = [1,2,3]
```

作成したDataFrameを表示すると、表形式になっていることがわかります（**図1**）。

```
In [5]: print(df)
```

```
      Name Country Age Birthday
1 Nakajima   Japan  51    11/20
2    Smith     USA  25     3/5
3     Chen   China  39     8/29
```

図1●DataFrameの表示



▶DataFrameをCSVファイルにする

PandasのDataFrameは、CSVファイルにエクスポートしたり、CSVファイルをインポートしたりできます。先ほどのDataFrameオブジェクトを、CSVファイルにエクスポートしてみましょう。

CSVファイルへエクスポートするには、to_csv関数を使います（**図2**）。

図2●to_csv関数の書式。引数は抜粋

```
DataFrame.to_csv(path_or_buf=None, sep=', ', na_rep='', float_for
mat=None, columns=None, header=True,  index=True, index_label=No
ne, mode='w', encoding=None, compression=None, quoting=None,  qu
otechar="", line_terminator='¥n', chunksize=None,
tupleize_cols=False, date_format=None,  doublequote=True,
escapechar=None, decimal='.')
```

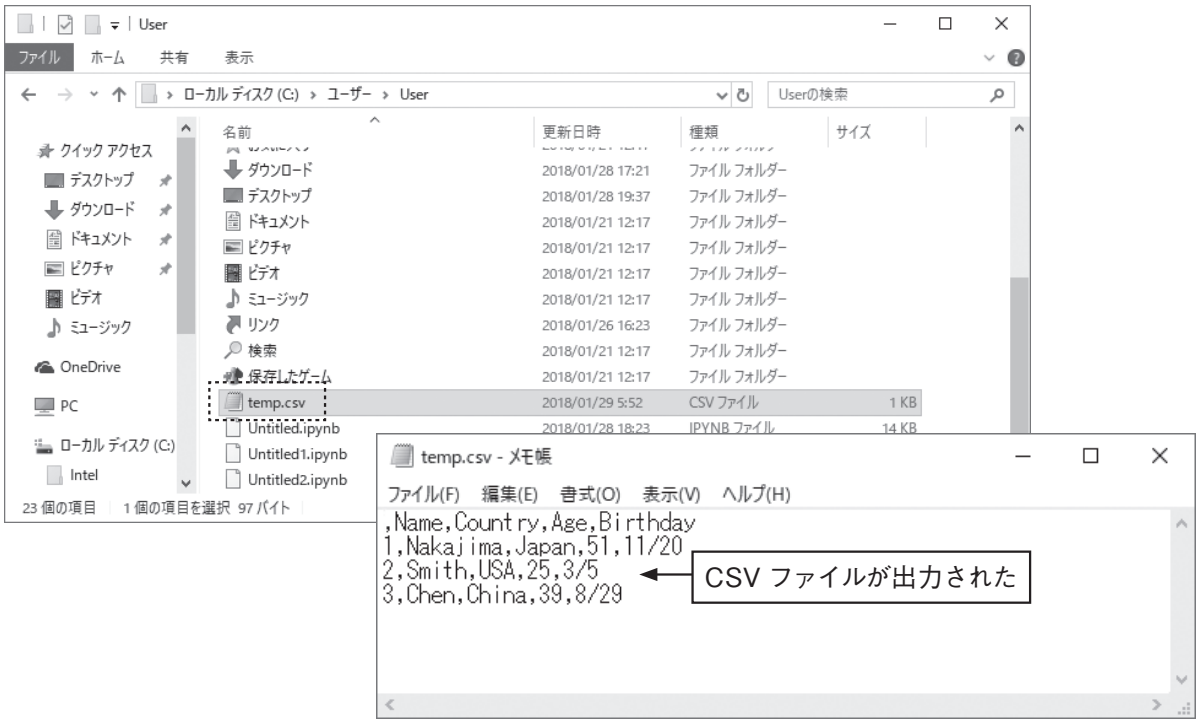
引数	説明
path_or_buf	出力するファイル名（省略した場合は、文字列として出力）
sep	区切り文字
float_format	浮動小数点数の書式文字列
columns	出力する列名
header	列名を保存するか否か
index	行名を保存するか否か
encoding	出力文字コード（'utf-8', 'shift_jis', 'euc_jp' 'ascii' など）
compression	ファイルの圧縮（'gzip'、'bz2'、'xz'）
line_terminator	改行文字
quotechar	引用文字
escapechar	エスケープ文字
date_format	日時の書式文字列
decimal	小数点の記号

temp.csvという名のファイルで保存してみます。

```
In [6]: df.to_csv("temp.csv")
```

実行すると、temp.csvというファイルができます。開くと、CSVファイルになっています（**図3**）。

図3●DataFrameをCSVファイルにする



▶CSVファイルを読みこむ

逆に、CSVファイルから、DataFrameを作ることもできます。temp.csvを次のように修正して保存します。

```
Name,Country,Age,Birthday
Nakajima,Japan,51,11/20
Smith,USA,25,3/5
Chen,China,39,8/29
```

CSVファイルからの読み込みは、read_csv関数を使います（**図4**）。

```
In [7]: dft = pd.read_csv('temp.csv')
```

エラーにならなければ、読み込めています。表示してみましょう。

```
In [8]: print(dft)
             Name Country  Age Birthday
        0  Nakajima   Japan   51    11/20
        1     Smith     USA   25     3/5
        2      Chen   China   39     8/29
```

図4●read_csv関数の書式。引数は抜粋

```
pandas.read_csv(filepath_or_buffer, sep=', ', delimiter=None, he
ader='infer',names=None, index_col=None,usecols=None, squeeze=Fa
lse,prefix=None, mangle_dupe_cols=True, dtype=None, engine=None,
converters=None, true_values=None, false_values=None,skipinitial
space=False, skiprows=None, skipfooter=None,nrows=None, na_value
s=None, keep_default_na=True, na_filter=True,verbose=False, skip
_blank_lines=True,parse_dates=False,infer_datetime_format=False,
keep_date_col=False, date_parser=None, dayfirst=False,iterator=F
alse, chunksize=None,compression='infer', thousands=None, decima
l='.', lineterminator=None,quotechar='"', quoting=0, escapechar=
None, comment=None, encoding=None, dialect=None, tupleize_cols=F
alse, error_bad_lines=True, warn_bad_lines=True, skip_footer=0,
doublequote=True, delim_whitespace=False,as_recarray=False, comp
act_ints=False, use_unsigned=False, low_memory=True, buffer_line
s=None,memory_map=False, float_precision=None)
```

引数	説明
filepath_or_buffer	入力用のファイル名（省略した場合は、文字列として出力）
sep	区切り文字
delimiter	sep の代わりとなる区切り文字
header	ヘッダー行の行数
names	ヘッダー行のリスト
index_col	行のインデックス番号
dtype	各行のデータタイプ（'a'、np.float64、'b': np.int32 など）
skiprows	先頭から読み込みをスキップする行数
skipfooter	末尾から読み込みをスキップする行数
encoding	入力文字コード（'utf-8', 'shift_jis', 'euc_jp' 'ascii' など）
quotechar	引用文字
escapechar	エスケープ文字
comment	コメント行の行頭文字
返り値	CSV ファイルを読み込んだ DataFrame オブジェクト

1日目 Part 6 matplotlib

科学技術計算などで、結果をグラフで表示できるとわかりやすい…、ということで、matplotlibというライブラリがあります。matplotlibを使えば、データを図にプロットできるようになります。matplotlibも、Anacondaに含まれています。

▶matplotlibのインポート

ここでは、pltという名でmatplotlib.pyplotをインポートします。また、グラフなどをJupyter Notebookのセルの下に直接表示するため、インラインの指定をします。

```
In [1]: import matplotlib.pyplot as plt

        #インライン表示するための宣言
        %matplotlib inline
```

▶グラフ表示

それでは、グラフを表示してみましょう。

matplotlib.pyplotモジュールのplot関数、show関数を使用します。使い方は、plot関数の引数でx軸、y軸を指定して、show関数で表示する手順です。x軸、y軸は配列やリストで渡します。

では、サイン波を表示してみましょう。サイン波のX座標はNumPyのlinspace関数で、Y座標はsin関数で計算します。それぞれの関数の書式は**図1**、**図2**のようになります。

mathモジュールのpiで円周率を取得し、X座標、Y座標を計算した結果をmatplotlibのplot関数に渡してグラフを生成し、show関数で表示します（**図3**）。

```
In [2]: import math
        import numpy as np

        x = np.linspace(0, 5 * math.pi)
```

図1●linspace関数の書式。引数は抜粋

```
numpy.linspace(start, stop, num = 50, endpoint = True, retstep
= False, dtype = None)
```

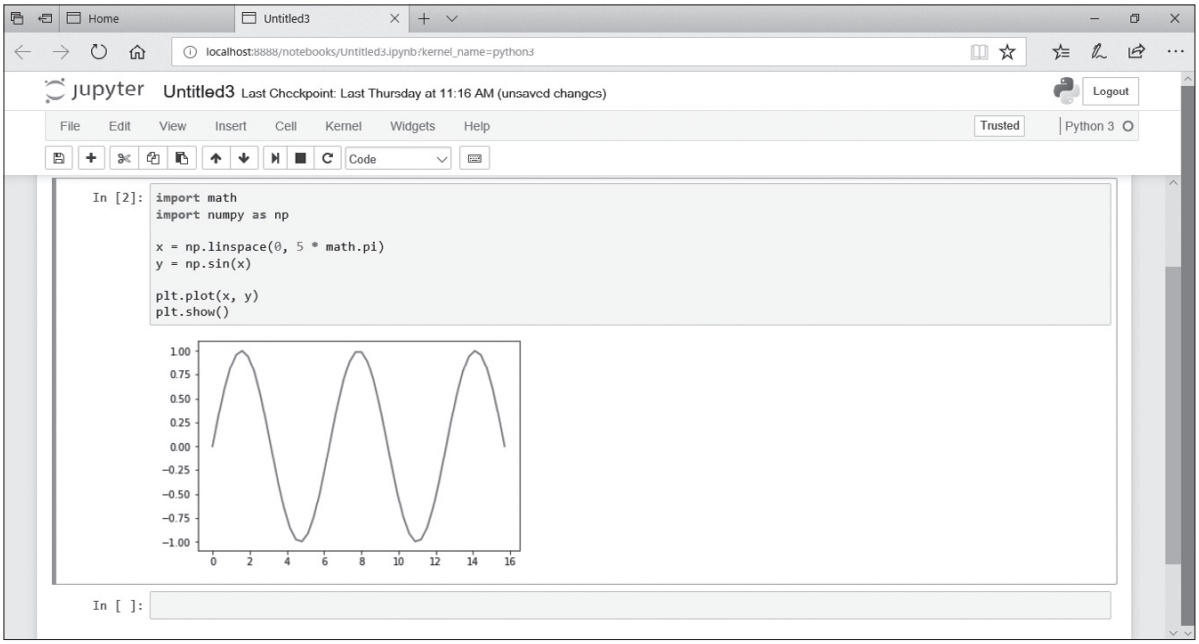
引数	説明
start	数列の始点
stop	数列の終点
num	生成するndarrayの要素数（デフォルトは50）
endpoint	生成する数列において、stopを要素に含むかどうか
dtype	出力するndarrayのデータ型を指定（ない場合float）
返り値	num等分された等差数列を要素とするndarrayオブジェクト

図2●sin関数の書式

```
numpy.sin(x[, out]) = <ufunc 'sin'>
```

引数	説明
x	ラジアン
返り値	三角関数サインの値

図3●サイン波の表示



```
        y = np.sin(x)

        plt.plot(x, y)
        plt.show()
```

▶タイトルや軸ラベルの設定

title関数でグラフのタイトル、xlabel関数、ylabel関数でグラフの軸ラベルを設定できます。グラフの凡例はplot関数の引数labelで凡例名を指定し、legend関数で表示します。

ちなみに日本語を表示するには、matplotlibの設定ファイルを書き換える必要があります。ここでは、グラフを確認できればよいので、設定ファイルの書き換えは行いません。

次のコードを入力して、グラフを表示してみましょう。

```
In [3]: # title関数でタイトルを追加
        plt.title('Sin Graph')

        # xlabel関数、ylabel関数で軸名を追加
        plt.xlabel('X-Axis')
        plt.ylabel('Y-Axis')

        # plot関数の引数labelで凡例名を指定
        plt.plot(x, y, label='sin')

        # legend関数でグラフの凡例を表示
        plt.legend()

        plt.show()
```

実行すると**図4**のようになります。

図4●タイトルや軸ラベルの追加