2-1 基本変数（数値／四則演算）

ここでは、整数、実数、文字列など、サンプルコードを実行しながら、Pythonの基本変数を習得していきます。

2-1-1 学習の前に

本格的なコードを書く前に、Pythonインタープリターを使って、Pythonの基本を学習します。本章では、対話形式でコードを実行していきます。QtConsole（1-3-5参照）、Jupyter Notebook（1-3-6参照）のいずれかを起動し、対話形式でコードを入力できるようにします（**図2-1**）。

Jupyter Notebookの場合、「In [1]」とプロンプトが表示されたら、Pythonコードを入力して実行します。なお、**本書ではプロンプトを「>>>」という表記で統一**しています。

図2-1 ▶ Jupyter Notebook画面

またWindowsの場合は「Anaconda Prompt(anaconda3)」、macOSの場合はターミナルを使ってサンプルコードを実行することもできます。

コマンドプロンプトやターミナルで「python」と入力すると、Pythonインタープリターを利用可能になります。「>>>」プロンプトに続けてPythonコードを入力して実行します。


```
> python ↵
Python 3.10.8 (tags/v3.10.8:aaaf517, Oct 11 2022, 16:50:30) [MSC
v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>> Pythonコードを入力して実行
```

2-1-2 基本的なデータ型

Pythonは、変数のデータ型を明示的に指定する必要がない**動的型付け言語**です。つまり、変数を使用する際にデータ型を意識する必要はありません。ここでは、データ型の基本として数値型と文字列型について解説していきます。

数値型

数値型は、演算子を使った計算や比較などが可能です。数値型にあたるデータ型として、整数だけを扱える**整数型（int型）**、小数点を含む数を扱える**浮動小数点型（float型）**、複素数を扱える**複素数型（complex型）**があります。

表2-1 ▶数値型

データ型	意味	例
int	整数	-2, -1, 0, 1, 2, 3
float	浮動小数点数	-1.234, 1.2e3 , 1.2E-3
complex	複素数	1j、1+2j、3-1j

文字列型

文字列型は、str型とも呼ばれる文字列を扱うためのデータ型です。演算子を使った計算や比較が可能です。また、**メソッド**を使って、文字列に対する操作を行うことができます。

Pythonの文字列型は、複数の文字を集めたデータとして定義されています。**インデックス**（要素の番号）や**スライス**（要素の範囲を指定すること）を使って、文字や部分文字列を取り出すことができます。

2-2-3 タプル型

タプル型は、リストと似た機能を持ったデータ型です。インデックスやスライスを使って要素を取り出すなど、リスト型と同じような操作が可能ですが、要素の変更はできません。その代わりに高速処理が可能なため、やりたいことがタプル型で事足りる場合は、リスト型を使わずにタプル型を使うようにしましょう。

タプル型は「( )」を使って定義します。

```
>>> tuple = ("a" ,"b" ,"c" ,"d" ,"e") ↵
>>> tuple ↵
('a', 'b', 'c', 'd', 'e')
```

リスト型と同じように要素を入れ替えようとするとエラーになります。

```
>>> tuple[2] = "B" ↵        リストと同じように要素を入れ替えようとすると
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment    エラーになる
```

2-2-4 セット型

セット型は、重複しない複数の要素を管理するために使用します。重複したデータを要素として持つことはできません。またリスト型とは異なり、順番という概念がないため、インデックスやスライスを使って各要素を取り出すことはできません。ただし要素の変更や追加・削除は可能です。

セット型は{ } を使って定義します。

```
>>> set = {"a" ,"b" ,"c" ,"d" ,"e"} ↵    セット型を定義
>>> set ↵
{'e', 'b', 'c', 'a', 'd'}                  順番は保持されない
>>> set[2] ↵    インデックスで要素を取り出そうとするとエラーになる
```



```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
>>> set.add("f") ↵        要素の追加
>>> set.remove("a") ↵     要素の削除
```

2-2-5 ディクショナリ型

ディクショナリ型は、**キー**と**値**をペアにして、複数の要素を管理するためのデータ型です。リスト型と同じく複数の要素を集めたデータ型ですが、順番という概念がありません。キーを利用して要素を取り出すことができます（他言語の連想配列やMapと呼ばれるデータ型と同じものになります）。

ディクショナリ型は「{キー:値, ……}」を使って定義します。キーを使うと、各要素の値にアクセスすることができます。

すべての要素を取り出すにはitems( )メソッド、すべてのキーを取り出すにはkeys( )メソッドを使用します。またすべての値を取り出すにはvalus( )メソッドを使用します。

```
>>> dict = {"Minato-ku":24.31,"Setagaya-ku":89.09} ↵
    ディクショナリ型を定義
>>> dict["Minato-ku"] ↵    キーを使って値を取り出す
24.31
>>> dict.items() ↵         すべての要素を抽出
dict_items([('Minato-ku', 24.31), ('Setagaya-ku', 89.09)])
>>> dict.keys() ↵          すべてのキーを抽出
dict_keys(['Minato-ku', 'Setagaya-ku'])
>>> dict.values() ↵        すべての値を抽出
dict_values([24.31, 89.09])
```

3-2 オブジェクトとクラス（マルチバイト文字列、ファイル入出力）

Pythonは、オブジェクト指向プログラミング言語の1つです。前章で解説した基本変数である文字列や、コレクションであるリストもオブジェクトとしての機能を持っています。ここでは、Pythonにおけるクラスの使用方法について解説します。

3-2-1 クラス

Pythonをはじめとする**オブジェクト指向**のプログラミング言語では、すでにあるオブジェクトに加えて、必要に応じてオブジェクトを作成しプログラムを作成します。その際、オブジェクトの性質や振る舞いをコードにしたものが**クラス**です。

```
class クラス名：
    クラスの内容
```

Pythonでは、「class クラス名:」と記述してクラスを作成します。慣習的に**クラス名の先頭は大文字**にします。**アトリビュート**（クラス内で定義された変数）や**メソッド**（クラス内で定義された関数）を記述する際は、インデント（スペース文字4つ分）を入れてください。

実際にクラスを使用したプログラムを例に解説します（**リスト3-2**）。

リスト3-2 ▶ sample3-2-1.py

```
01 class MyClass:
02
03     value = 100              アトリビュートの定義
04
05     def getValue(self):      メソッドの定義
```



```
06         print(self.value)
07
08     def addValue(self):      メソッドの定義
09         self.value += 1
10
11 myclass = MyClass()
12 myclass.addValue()
13 myclass.getValue()
```

value = 100ではアトリビュートを定義し、def getValue( )やdef addValue( )ではメソッドを定義しています。

メソッドを定義する際は「**def メソッド名(引数...):**」と記述します。アトリビュートもメソッドもクラス定義ブロックの中（インテンドで字下げされた）にあることに注意しましょう。

リスト3-2を実行すると、次のように表示されます。

```
> python sample3-2-1.py
101
```

リスト3-2では、クラスを定義している以外に次の処理も行われています。

```
11 myclass = MyClass()    変数「myclass」にMyClassインスタンスが生成され代入される
12 myclass.addValue()
       MyClass( )のaddValue( )メソッドを実行（アトリビュートvalueに1が足される）
13 myclass.getValue()
       MyClass( )のgetValue( )メソッドを実行（アトリビュートvalueを表示）
```

ここで気になるのが、クラス定義内でメソッドを定義する際に用いられている引数の「self」です。メソッドを定義する際には「self」が用いられているのに、実際にクラスをインスタンス化しメソッドを呼び出す際には、何も引数を指定していません。

これは他のプログラミング言語では見られないものですが、Pythonではメソッドを定義する際は、必ず1つ以上の引数を持つようにし、**第1引数に自分自身の意味を持つ「self」を受け取るようにします。**

Pythonはメソッドを呼び出す際に、暗黙的に第1引数としてインスタンスを渡しているため、明示的にselfを引数に渡す（たとえば、myclass.getValue(myclass)とする）必要はありません。

3-4 正規表現

ここでは、Pythonで正規表現を使用する方法について解説します。文字列オブジェクトには、文字列を操作するメソッドがありますが、単純な文字の組み合わせしか検索や置換の条件に指定することができません。大文字／小文字、部分一致など、検索条件を柔軟に組み立てる際には、正規表現を使用します。

3-4-1 Pythonでの正規表現の利用

3-3で解説した文字列オブジェクトは、strip( )／split( )／replace( )などのメソッドを使って文字列を操作できますが、置き換える文字列や区切り文字など、単純な文字列しか引数に指定できません。

たとえば、split("a")は「a」にマッチした個所で文字列を分割しますが、大文字の「A」にはマッチしません。よって大文字の「A」をマッチさせたい場合は、string.split("A")を実行する必要があります。

また、「string.replace("AD2017","西暦2017年")」は、「AD2017」を「西暦2017年」に置き換えることはできますが、「AD1970」を「西暦1970年」に置き換えるようなことはできません。

このような「a」にも「A」にもマッチさせたい場合や、「AD<数字4桁>」を「西暦<数字4桁>年」に置きかえたい場合など、柔軟性の高い文字列パターンを指定するには、**正規表現**を用います。

正規表現とは、文字列の並びやパターンを表現するものです。Pythonに限らず、他のプログラミング言語でも利用されており、**メタ文字**（プログラム上、特殊な意味を持つ文字）や「\（バックスラッシュ）」で始まる**特殊シーケンス**を使って表現します。なお「\（バックスラッシュ）」は、Windowsキーボードでは「\（円）」キーを、Macキーボードでは「option」キーを押しながら「¥」キーを押すことで入力できます。

3-4-2 正規表現を使ったPythonスクリプトの例

Pythonで正規表現を使用する場合は、**reモジュール**を使用します。モジュールについては4章で解説しますが、ここでは「モジュールを使用する際はプログラムのソースコード1行目にimport reと追加する」とだけ押さえておきましょう。

```
01 import re    reモジュールの読み込み
```

reモジュールを読み込むことによって、正規表現を使用した文字列の検索／置換／連結／分割などの処理が可能になります。具体的な使用例を見ていきましょう。

```
01 src_str = "abcdefgABCDEFG"  置き換える前の文字列
02 dst_str = src_str.replace("a","1")
03 print(dst_str)
```

上記例では、置き換える対象である文字「a」とそれが置き換えられる文字「1」を引数に、文字列オブジェクトre.replace( )メソッドを実行しています。これを実行すると、次のように表示されます。

```
1bcdefgABCDEFG
```

「a」が「1」に置き換わっていますが、大文字である「A」はマッチしないためそのまま残っています。

次はreモジュールを読み込み、正規表現を使って文字列の置き換えをしてみましょう。正規表現を使って、「a」と「A」を「1」に置き換える具体例を見てみましょう。

リスト3-8 ▶ sample3-4-1.py

```
01 import re                          reモジュールの読み込み
02
03 src_str = "abcdefgABCDEFG"        置き換える前の文字列
04 pattern = r"[aA]"                  置き換えられる文字列のパターン
05 dst_str = re.sub(pattern, "1", src_str)
```

4-3 モジュールの基本

ここでは、モジュールを読み込む方法やモジュールを作成する方法など、Pythonにおけるモジュールの使い方について解説していきます。

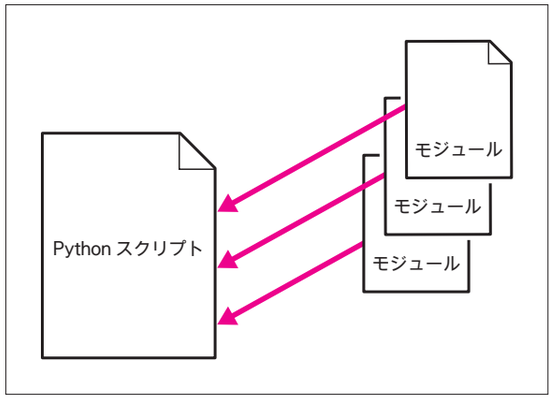
4-3-1 モジュールとは

モジュール（module）とは、関数を定義しているコードをファイルとして保存し、他のPythonスクリプトから利用できるようにしたものです（**図4-1**）。

一度定義した関数を、別のPythonスクリプトでも利用したい場合、コピー＆ペーストで毎回コードを貼り付けなくても、関数を定義しているコードをファイルに保存し、実行したいPythonスクリプトファイルから読み込めるようにしたのがモジュールで、これを利用することで再利用性が高まり、コーディングを効率よく進めることができます。

また自身で定義した関数以外にも、Pythonには便利な関数が多数用意されており、公開されているモジュールを読み込むことで、そうした関数を利用できるようになります。

図4-1 ▶ モジュール



4-3-2 import文

Pythonでモジュールを読み込むには、**import文**を使用することは3-4で解説しました。モジュールを読み込むには、importの後にモジュール名を指定します。

```
import モジュール名
```

リスト4-9では、timeモジュールを読み込み、timeモジュールに含まれるsleep( )関数を使えるようにするスクリプトです。sleep( )関数を実行すると、引数に指定した秒数の間、処理を一時的に停止します。

リスト4-9 ▶ sample4-3-1.py

```
01 import time
02
03 print("プログラムスタート")
04 time.sleep(5)
05 print("5秒経過")
```

リスト4-9を保存して実行すると、次のように表示されます。

```
> python sample4-3-1.py
プログラムスタート
5秒経過
```

5秒経ってから表示される

モジュールに含まれる関数を呼び出すには、モジュール名に続けて「.（ドット）」と**関数名**を記述します。リスト4-9では、timeモジュールに含まれるsleep( )関数を呼び出す際に、time.sleep(5)と記述しています。

```
モジュール名.関数名(引数1, 引数2,...)
```

3-4においても、import reでreモジュールを読み込んだ後、re.split( )やre.sub( )のように、「モジュール名.関数名( )」とすることで、関数を呼び出していたこと

5-1 Webアプリケーションの仕組み

ここでは、Webアプリケーションの仕組みやデザインパターン、データベースについて理解を深めていき、最後に本書で利用するWebアプリケーションフレームワークDjangoの特徴などについて解説していきます。

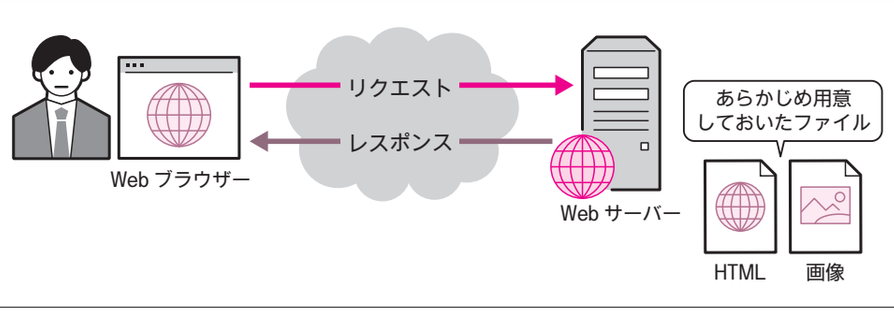
5-1-1 静的／動的コンテンツとは

サーバーがWebブラウザーからリクエストを受け取ったあと、サーバー内部で実行される処理は、大きく静的コンテンツと動的コンテンツの2種類に分けることができます。

静的コンテンツ

1つはクライアントからのリクエストに応じて、あらかじめ用意された画像やHTMLドキュメントをレスポンスとして送信するものです。このようなコンテンツを**静的コンテンツ**と呼びます。リクエスト内容に応じてまったく同じページが表示されるため、いつ／誰がアクセスしても同じページが表示されます（**図5-1**）。

図5-1 ▶静的コンテンツ

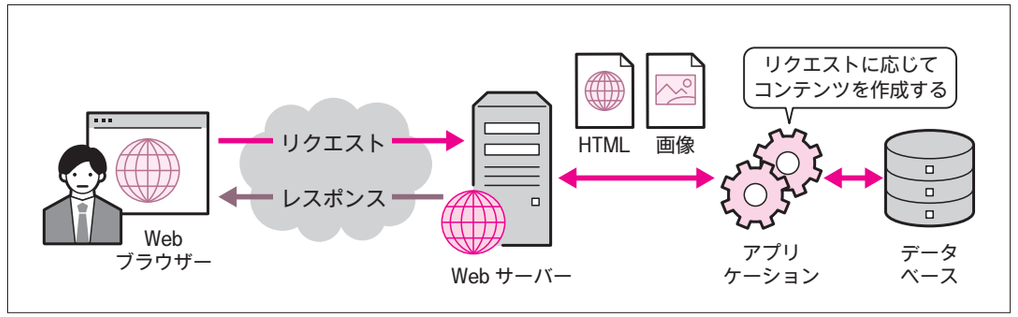


動的コンテンツ

もう1つは、リクエストの内容に応じて、HTMLドキュメントなどのコンテンツを自動的に生成し、それをクライアントにレスポンスとして送信するものです。この自動的に作成されるコンテンツを**動的コンテンツ**と呼びます（**図5-2**）。

現在では、Webサーバーで扱われているWebコンテンツの多くは動的コンテンツです。たとえば、ニュース配信サイトで表示される「本日の注目記事」や、ショッピングサイトで表示される「あなたへのオススメ商品」などが身近な動的コンテンツの例です。これらはアクセスするユーザーの情報やアクセス履歴などを解析し、自動的に作成された動的コンテンツとなります。

図5-2 ▶動的コンテンツ



5-1-2 動的コンテンツが作成される仕組み

5-1-1では、静的コンテンツと動的コンテンツの違いについて解説しました。静的コンテンツは、あらかじめ決まったコンテンツを用意しておけばよいですが、動的コンテンツの場合は、コンテンツを作成する仕組みが必要となります。そのために必要な仕組みとして、主に次の2つが必要となります。

- **データを保存する仕組み**
- **サービスを実現するためのロジック**

データを保存する仕組み

コンテンツを作成するには、元となるデータが必要となります。ユーザー情報／商品情報／

5-3 Djangoによる Webアプリケーションの作成

ここまででDjangoのインストールが完了し、Webアプリケーションを開発する環境が整いました。それではDjangoでかんたんなアプリケーションを作成してみましょう。

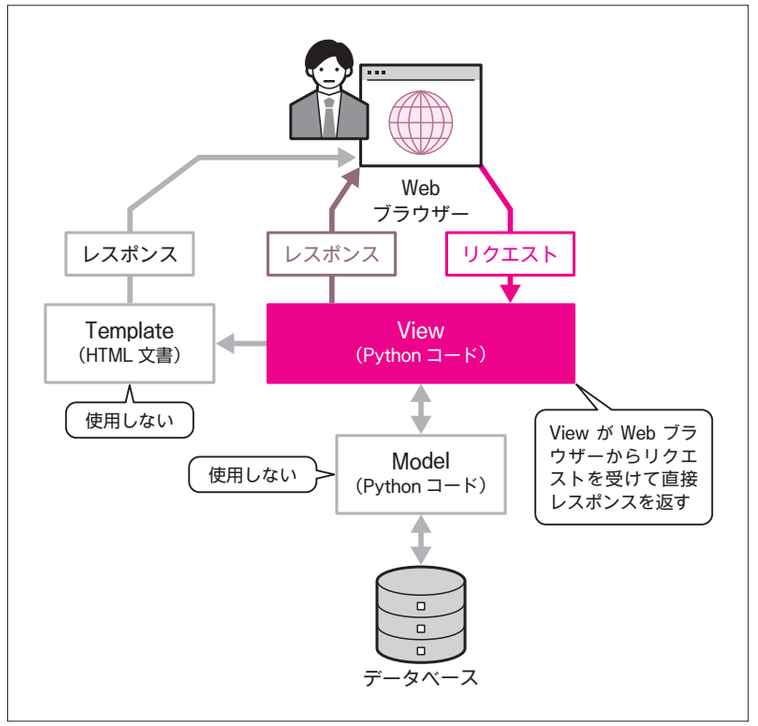
本節では、**図5-6**のように表示されるWebアプリケーションを作成していきます。

図5-6 ▶作成するWebアプリケーションの実行イメージ

5-3-1 作成手順

まずDjangoを使った開発の流れを理解するため、データベースやテンプレートエンジンを使用しません。MTVアーキテクチャのModelやTempleを使用せずに、Webアプリケーションを作成します（**図5-7**）。

図5-7 ▶今回作成するWebアプリケーションのアーキテクチャ



次の手順でDjangoを使ってWebアプリケーションを開発します。

①Djangoプロジェクトの作成
②設定ファイルの修正
③Webアプリケーションの作成とDjangoプロジェクトへの登録
④Viewの定義
⑤URLディスパッチャーの作成
⑥開発サーバーの起動
⑦動作確認

5-3-2 作業上の注意

DjangoでWebアプリケーションを作成する際は、次の点に注意してください。

6-3 NumPyの利用

6-2までででデータサイエンスの概要と、Pythonでデータサイエンス行うのに必要なライブラリーを紹介しました。本節では、数学関数(線形代数)ライブラリーであるNumPyの使い方について解説していきましょう。

6-3-1 本節で学習すること

本節では実際にNumpyを操作していきます。次の内容について理解を深めていきます。

- **NumPyの基礎(6-3-2)**
- **配列の生成(6-3-3)**
- **配列の形状の変換(6-3-4)**
- **配列のprint(6-3-5)**
- **配列の演算(6-3-6)**
- **配列のインデックス参照、スライス(6-3-7)**
- **ファイルの入出力(6-3-8)**

Windowsでは「Anaconda Prompt(anaconda3)」、macOSでは「ターミナル」を開いて、コマンドを入力できるようにしてください。プロンプトが表示されたらPythonインタープリターを起動します。「>>>」プロンプトが表示され、Pythonコードを入力できるようにします。

```
> python ↵   Pythonインタープリターの起動
Python 3.9.12
省略
>>>          Pythonコードを入力
```


6-3-2 NumPyの基本操作

NumPyが扱う多次元配列に格納されるオブジェクトは同じ型のみになります。通常は整数や実数といった数値型オブジェクト(データ)が格納されます。NumPyにおいて、次元のことを**軸(axis)**、軸の数のことを**ランク(rank)**と呼びます。たとえば、次の配列はランク2(2次元)になります。

```
[[ 1., 3., 2.],
 [ 1., 1., 1.]]
```

NumPyの配列クラスは**ndarray**と呼ばれています。arrayという別名が使われることもあり、本章でもクラス名として「array」を使用します。標準Pythonライブラリーの`array`クラスと同名ですが、NumPyの配列クラスでは多次元配列が扱うことができるなど、大きく異なります。混同しないようにしましょう。

コードを実行する前に、NumPyライブラリーをインポートします。NumPyの各関数を簡単に呼び出せるよう「import … as …」構文(4-3-4参照)を使って、次のようにインポートを実行します。

```
>>> import numpy as np ↵
>>>
```

これで「np.関数名()」でNumPyの関数を呼び出すことができるようになりました。

6-3-3 配列の生成

NumPyで用意されている各種関数を使って、新規に配列を作成する方法を解説します。

NumPyは、CSVファイル(データの項目間がカンマで区切られている)やTSVファイル(データの項目間がタブで区切られている)から、データを読み込んで配列として格納することができます。読み込む方法については、6-3-8で解説します。

まずは新規に配列を作成してみましょう。以下の内容でNumPyの配列(ndarrayまたはarray)を作成する方法を解説します。

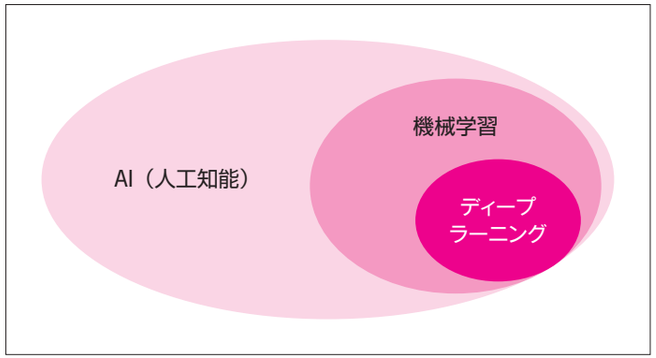
7-1 AI／機械学習／ディープラーニングとは

本節では、AI／機械学習／ディープラーニングについて、それぞれの概要や関係性について解説します。

7-1-1 AI／機械学習／ディープラーニングの関係

AI（Artificial Intelligence）とは、人間の学習と同様の機能を機械であるコンピューターで実現したシステムのことです。**機械学習（マシンラーニング）**はAIの一分野であり、さらに、**ディープラーニング（深層学習）**は、機械学習の一分野という位置付けとなります（**図7-1**）。

図7-1 ▶ AI／機械学習／ディープラーニングの関係



AIは人間の「知能」（学習／推論／判断）を機械で人工的に再現したコンピューターシステムです。その歴史は古く、1956年のダートマス会議で「AI（Artificial Intelligence）」という言葉が使われ、研究分野として確立されていました。

AIと呼称されているものは多数あります。たとえば、日本語入力のAI変換のように、人間が持つ知識をコンピュータに登録して回答を導くエキスパートシステムや、Siri、Cortanaなどの音声アシスタントシステム、自動運転システム、ロボット掃除機、チェスプログラムのディープブルー、囲碁プログラムのAlphaGOなど、多岐にわたります。

これまでAIは、狭い領域で成果を上げることはできても、広い領域での応用が利く人間の「知能」には及ばないといったことが繰り返し述べられてきました。そんな中、機械学習が実用化されたことで、AI研究は大きく飛躍しました。

機械学習とは、大量の学習データを解析し、規則性や関係性を見つけ出す手法です。機械学習を利用することで、分類や区別（**学習**）／予測（**推論**）／判断を行うことが可能になります。

理論自体は以前から存在していたものの、コンピューターの性能が不十分だったり、計算コストが高かったりと、十分な成果を上げられませんでした。しかし、近年コンピューターの性能が飛躍的に向上し、また手法が進展したことで実用性が格段に向上しました。　機械学習によって規則性や関係性を見つけ出すためには、データを比較する際に着目すべき特徴の組み合わせである**特徴量**を、人間が見つけ出して設定する必要があります。特徴の組み合わせである特徴量の設定次第で結果が大きく異なるため、結局は扱う人間に大きく依存することになります。このような問題を解決する手段として、ディープラーニング（深層学習）が注目されています。

人間の脳の働きについての研究が進み、その成果を応用した機械学習手法の1つとして考案されたのがディープラーニングです。**特徴量の選定や組み合わせを、データを解析することで自ら作り出すことができる**ため、人間への依存度は小さくなります。また、データ量を増やすことによって、規則性や関係性を見つけ出す精度も向上するメリットもあります。ディープラーニングはAI研究において注目され、第三次AIブームの火付け役となりました（**図7-2**）。

図7-2 ▶ AI／機械学習／ディープラーニングの特徴

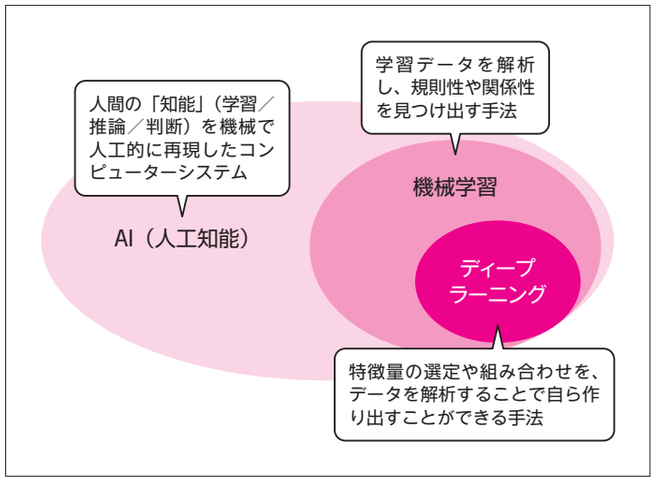


図7-40 ▶ Sequentialモデル

```
In [5]: model = tf.keras.models.Sequential([
          tf.keras.layers.Flatten(input_shape=(28, 28)),
          tf.keras.layers.Dense(128, activation='relu'),
          tf.keras.layers.Dropout(0.2),
          tf.keras.layers.Dense(10)
        ])
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

最初の層は、「tf.keras.layers.Flatten(input_shape=(28, 28))」になります。28×28の2次元データを`Flatten( )`関数によって、1次元配列に変換したものを最初の入力層に使用しています。

「tf.keras.layers.Dense(128, activation='relu')」は、入力層につながる次の層（中間層）になります。128ノードからなり、活性化関数`relu`を用いて、入力値が0以下なら「0」を、0より大きければ入力値そのものを返します。

「tf.keras.layers.Dropout(0.2)」は、直前の層に対して、ドロップアウト率（0.2）を設定して過学習を防ぎ、モデルの精度を向上するようにしています。

最後の「tf.keras.layers.Dense(10)」は出力層になり、0～9のどの数字かを判定するため、出力層は10ノードになります。

④モデルの設定

予測値と正解値の誤差を計算する損失関数を定義します（**図7-41**）。チュートリアルでは損失関数として、**交差エントロピー**を使用しています。

図7-41 ▶損失関数の定義

```
In [6]: loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

モデルのパラメータを設定して学習方法を定義します（**図7-42**）。必要なパラメータは、最適化関数、損失関数、メトリクスの3つになります。

最適化関数には、収束が速いとされる「Adam（Adaptive Moment Estimation）」を設定しています。損失関数には、先ほど定義した「交差エントロピー」を設定します。メトリクスとして、手書き文字の分類では定番の「accuracy」を設定します。

図7-42 ▶学習方法の定義

```
In [7]: model.compile(optimizer='adam',
                      loss=loss_fn,
                      metrics=['accuracy'])
```

```
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
```

⑤モデルの学習

`fit( )`関数を実行してモデルの学習（トレーニング）を行います（**図7-43**）。引数には、トレーニングデータ、教師ラベルデータ、エポック数を指定します。エポック数によって、何回繰り返して学習させるかを決めます。チュートリアルではエポック数を「5」と設定しているため、5回学習が行われます。

図7-43 ▶モデルの学習

```
In [12]: model.fit(x_train, y_train, epochs=5)
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0422 - accuracy: 0.9862
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0404 - accuracy: 0.9869
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0387 - accuracy: 0.9873
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0368 - accuracy: 0.9873
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0328 - accuracy: 0.9885
Out[12]: <keras.callbacks.History at 0x7f994cc29100>
```

```
model.fit(x_train, y_train, epochs=5)
```

⑥モデルの評価

最後に学習したモデルに対して評価（テスト）を実施します（**図7-44**）。トレーニングに利用