Pythonが5日でわかる本

パイソン

基本編

中島 省吾（メディアプラネット）著

人気No.1
プログラミング言語
「Python」の基本を
学ぶ新人研修を
誌上体験しよう

日経ソフトウエア
NIKKEI SOFTWARE
2019年9月号 第2付録

特別付録② Pythonが5日でわかる本 基本編

日経ソフトウエア
NIKKEI SOFTWARE

1日目

Pythonの概要と環境構築

1日目 Part 1 はじめに

ここは、とある大手水産会社の本社第七会議室。会社は、魚の養殖事業に流行りのAI（人工知能）やIoTを活用したいということで、水産学部と理学部生物学科出身だが、数学が比較的得意な新人2人、小鯖 進（こさば すすむ）と栄井 愛（さかい あい）を、「AI人材」として育成することにした。その最初のステップがプログラミング言語「Python」の習得。講師と合わせて3人だけの新人研修が始まる…。

進 おや、栄井さん…、愛ちゃんじゃがね? おはようさん。

愛 おはよう…。

進 愛ちゃんも、Pythonの研修に行くように言われたん?

愛 …小鯖君も、数学のテスト、成績良かったのね。

進 たまたま知っとる分野が出ただけじゃな。たまたまだよ、たまたま!

講師 ハイ、それでは始めます。おはようございます。今日から、Pythonプログラミングの学習をお手伝いする講師の中島です。

進 **愛** おはようございます。

講師 さっそくお二人に質問ですが、Pythonというプログラミング言語について、何か知っていることはありますか?

進 そうやのう、最近はAIプログラミングで注目されていると新聞に書いてあったのう。ITの専門家でなくても使いやすいとも書いてあった。

愛 Pythonは、オランダのグイド・ヴァンロッサムが開発した、オブジェクト指向スクリプト言語。名前の由来はイギリスのBBCが製作したコメディ番組「空飛ぶモンティ・パイソン」。

進 なっ! よう知っとるのう。

講師 お二人とも、よく知っていますね。そうですね、ほかのプログラミング言語に比べて文法が簡単で、学習しやすいという特徴を持っていますね。少し専門的に言うと、「インタプリタ」で実行される「動的型付き言語」という特徴を備えています。

進 インタプリタ? 動的…何?

講師 プログラミング言語は、大きく「コンパイル言語」と「インタプリタ言語」に分けることができます。コンパイル言語は、プログラムのコードを機械語に翻訳してから実行しますが、インタプリタはプログラムのコードを1行ずつ翻訳しながら実行します。ただ、Javaというプログラミング言語のように、コンパイルして「バイトコード」を生成後に、そのバイトコードを1行ずつインタプリタで実行するハイブリットな言語もあります。

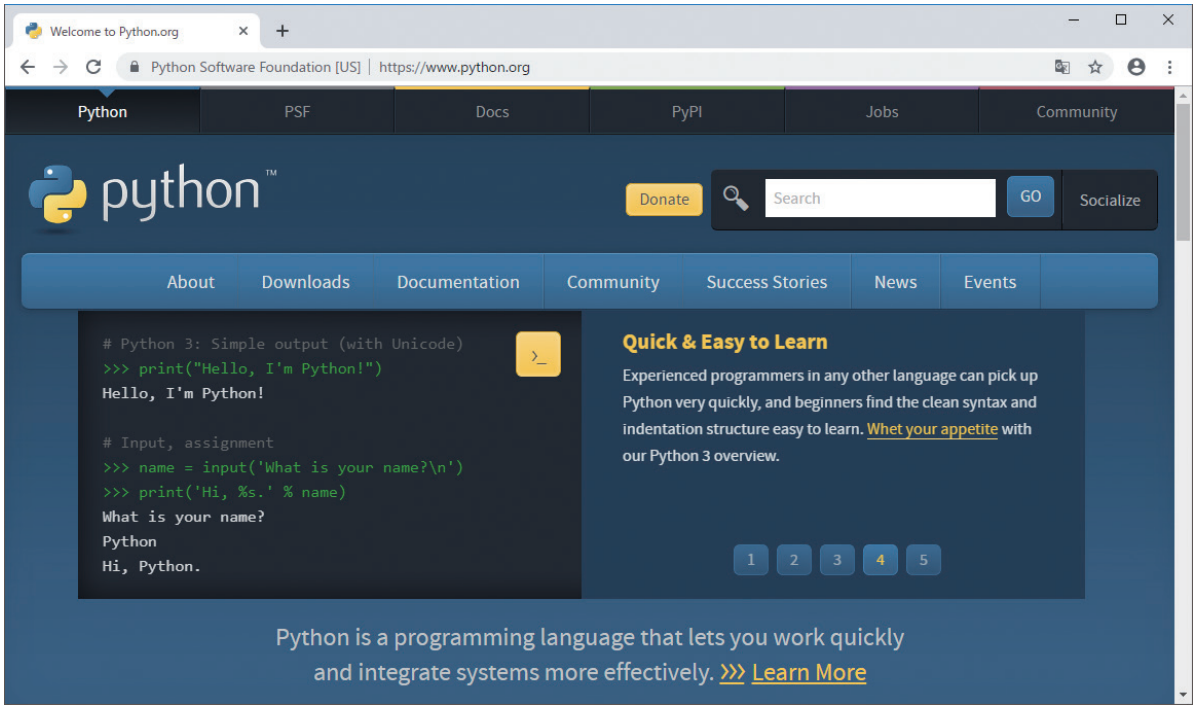
愛 ハイブリットの方がいい…。

進 とはいっても、Pythonを学ぶ研修だしのう。

講師 Javaもすばらしいプログラミング言語ですが、「静的型付き言語」なので「データ型」、つまり文字列型や整数型といったデータの種類を強く意識しながらプログラミングする必要があり、難易度は若干高くなります。Pythonは、動的型付けが可能なので、データ型は実行中に決まります。言い換えると、それほど強くデータ型を意識しなくてもよいので、手軽にプログラミングができます。

進 Python、やるな。

図1●Pythonの公式サイト（https://www.python.org/）



講師 また、Pythonのコードは誰が書いても同じようになる傾向があります。そのため、可読性に優れています。読みやすいということはわかりやすいということですから、プログラミング教育の分野でも、大注目の言語なんです。それでは、お二人のパソコンを起動してください。起動したら、Pythonの公式サイト（https://www.python.org/）にアクセスしてみましょう（**図1**）。

愛 ダークなサイト…。

講師 公式サイトでは、Pythonのコードを直接入力して試すことができます。まず、トップページのコードが並んだ部分にある「Launch Interactive Shell」（インタラクティブシェルを起動する）のボタンをクリックしてください。しばらくすると、「インタラクティブシェル」（Interactive Shell）と呼ばれる画面が表示されます（**図2**）。

講師 それでは、「>>>」の後ろに「print('Hello Python!')」と入力して、[Enter]キーを押してみてください（**図3**）。

図2●Launch Interactive Shell

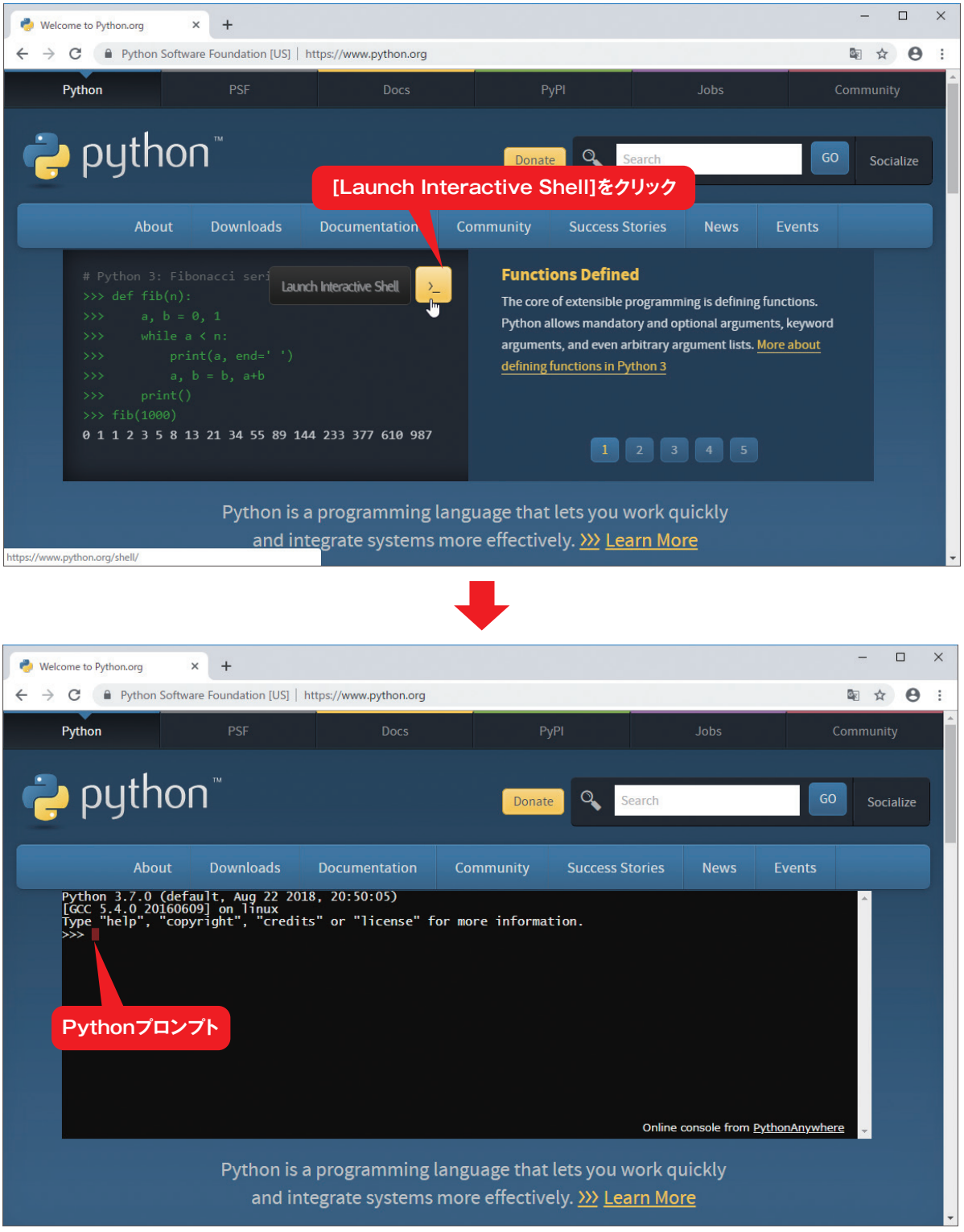
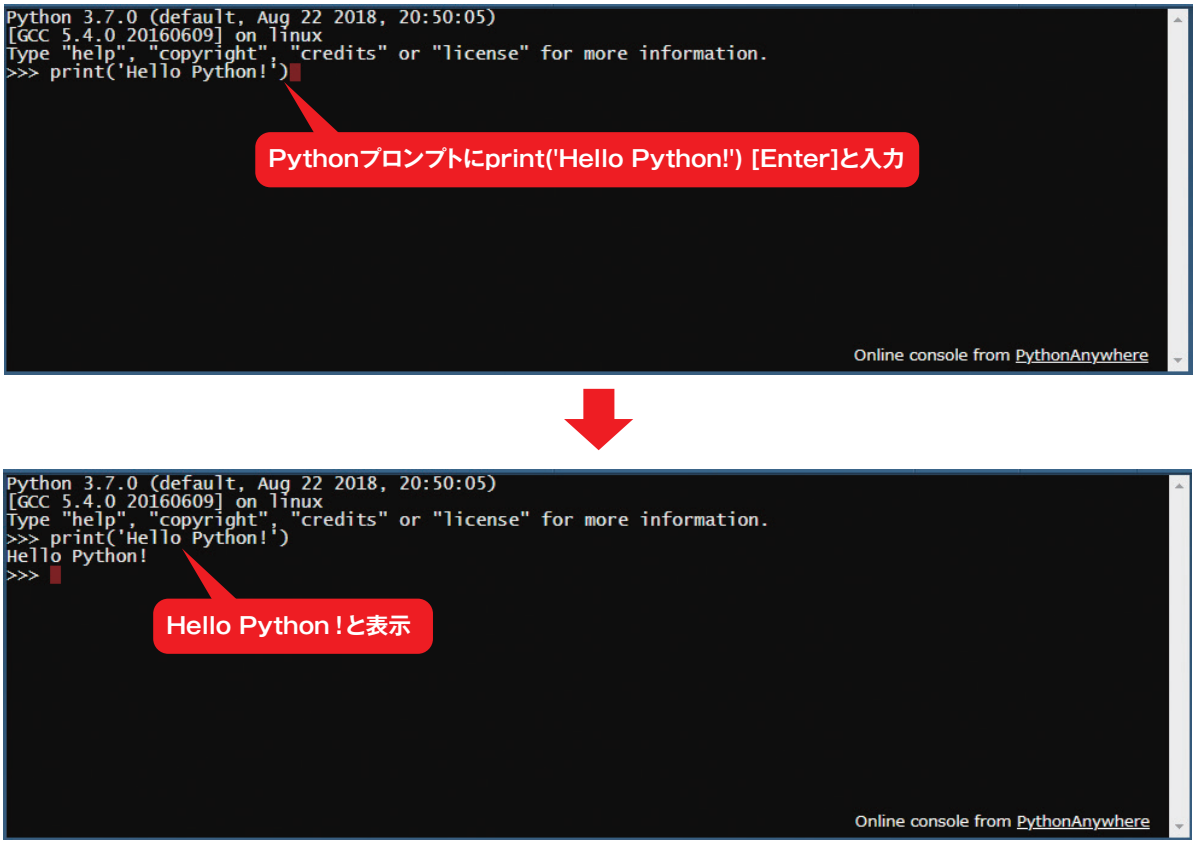


図3●初めてのPythonプログラム



愛 ハロー…。

講師 「>>>」の記号を、「Pythonプロンプト」と呼びます。この後ろにPythonの命令、つまりプログラムのコードを入力して実行することができます。このようにキーボードで入力と実行を行う画面を「端末」とか「コンソール」と呼ぶことがあります。文字列を表示するprintという命令は、「関数」と呼ばれるものです。print関数は、続く括弧の中の値をコンソールに表示します。

進 それじゃあ、このインタラクティブシェルを使ってPythonを学ぶのかのう。

講師 いえ、今回の研修では、公式サイトのインタラクティブシェルは使いません。「Anaconda」（アナコンダ）と呼ばれるPythonのパッケージを使います。

進 なんだか、たくましいのう。

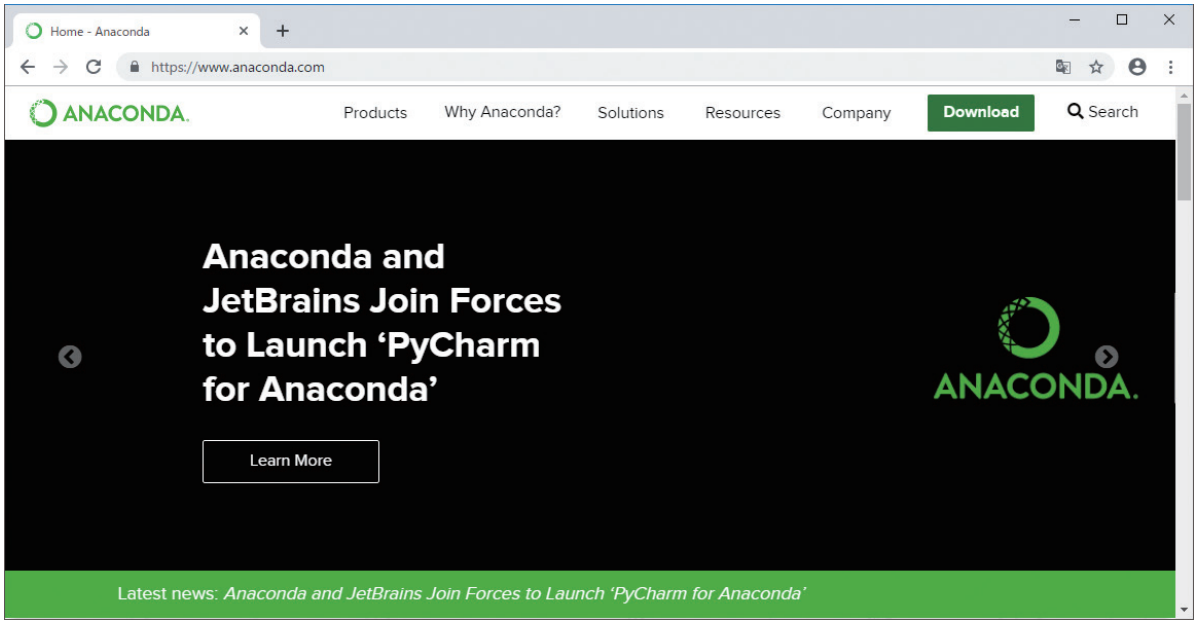
愛 アナコンダ…ステキな名前…。

講師 …。いったん、休憩を入れましょう。次は、Anacondaのインストールを行います。

Part 2 Anacondaのインストール

講師 それでは、Anacondaをインストールしましょう。まず、Anacondaのサイト（https://www.anaconda.com/）にアクセスします（図1）。

図1●Anacondaの公式サイト（https://www.anaconda.com/）



講師 サイトにアクセスしたら、メニューに「Download」ボタンがあるのでクリックします。表示からわかるように、Pythonのバージョンは2.xと3.xの2種類あります。もし、すでにある2.xで構築されたプログラムをメンテナンスする場合は、2.xを選択します。それ以外の場合は、2.xのサポートは2020年に終了するので、3.xを選びましょう。OSは、インストール先

のパソコンに合わせます。お二人は、Windows 10の64bit版をダウンロードしてください。

進 「Anaconda3-2019.03-Windows-x86_64.exe」ってファイルをダウンロードできたのう。

講師 そのファイルはインストーラーで、ファイル名の「2019.03」は、バージョンを表す日付です。インストーラーをダウンロードできたら、実行してください（**図2**）。インストーラーが起動するので、[Next]ボタンで次へ進みます。ライセンス承諾画面では[I Agree]を選択します。以降、すべてデフォルト設定のまま、[Next]ボタンで進んでかまいません。

講師 インストール後に、おすすめの開発環境を紹介する画面が出る場合は、使用しないので[Next]ボタンで進みます。最後に［Finish］ボタンが表示されたら、Anacondaの起動チェックを2つとも外して［Finish］ボタンをクリックします（**図3**）。

講師 これで、Anacondaをインストールできました。Anacondaをインストールすると、いくつかPythonの開発環境がインストールされます。その中から、今回は「Spyder」（スパイダー）というツールを使います。

進 Pythonって、蛇とか蜘蛛とか、変わっとるのう。

愛 蜘蛛…。

講師 では、Spyderを起動してみましょう。Windowsなら、スタートボタンをクリックして表示されるメニューの中に「Anaconda3(64-bit)」メニューがあるので、その中から「Spyder」を選びましょう。しばらくすると、「Windows セキュリティの重要な警告」というウインドウが表示されるので[アクセスを許可する]を選択すると、Spyderが起動します（**図4**）。

進 おお、たくさんボタンがあるのう。正直、難しそうだな。

講師 Spyderには多くの機能がありますが、よく使う機能は限られています。左側の

図2●インストーラーをダウンロードして起動

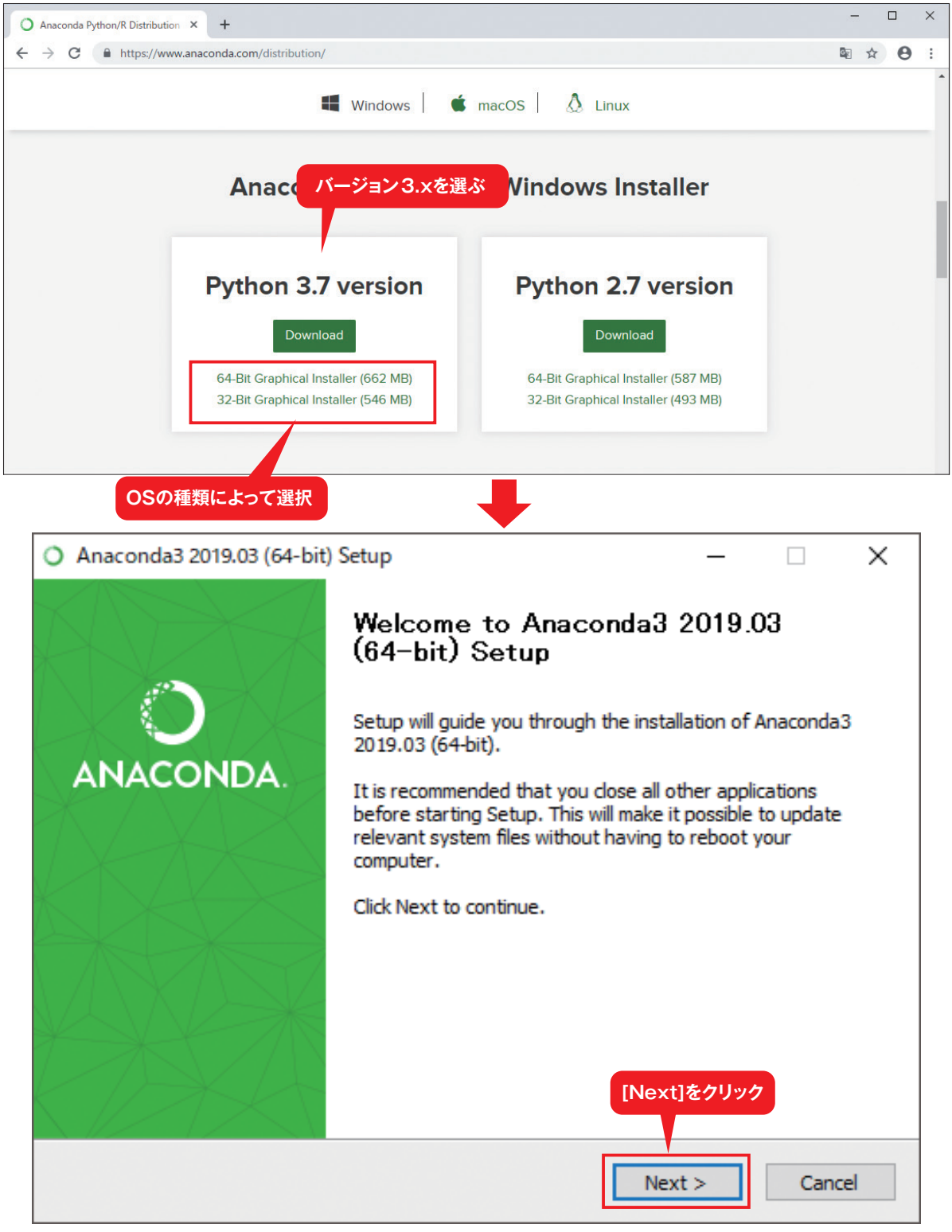


図3●Anacondaのインストールが完了

領域は、テキストエディターです。このエディターは、明日使います。今日は、右下にある「IPythonコンソール」を使います。表示されている「In [1]:」が入力プロンプトです。**図5**

のように、コードを入力してみましょう。

愛 公式サイトのインタラクティブシェルと同じ…。

講師 それでは、このSpyderを使ってPythonの学習を進めましょう。

図4●Spyderを起動

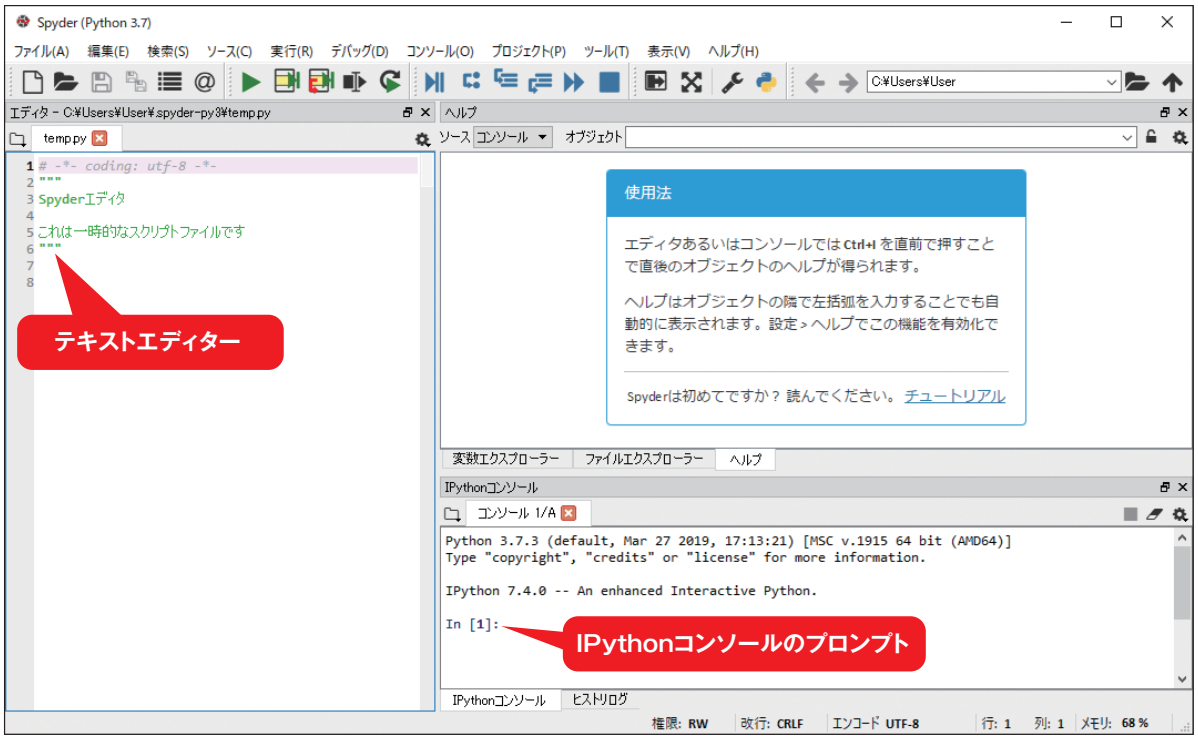
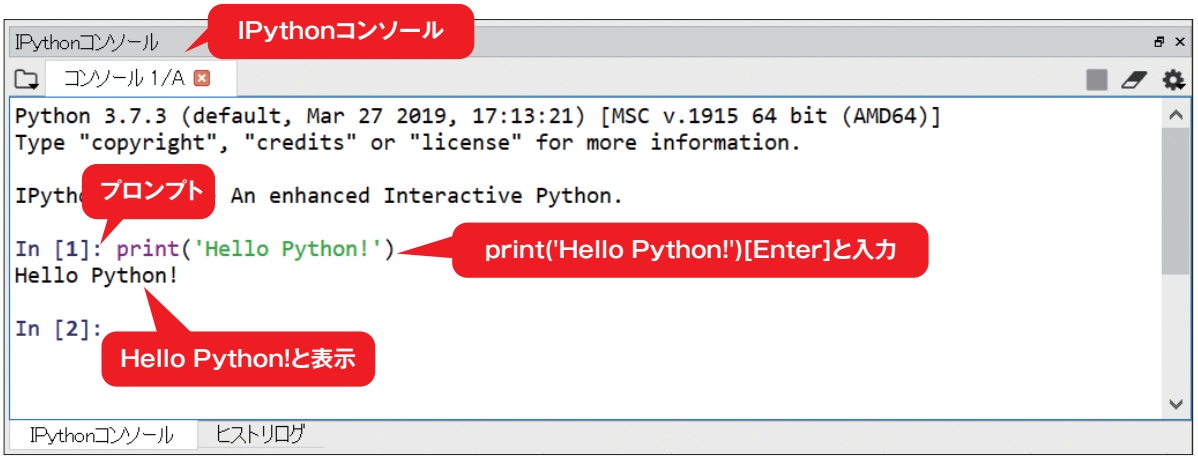


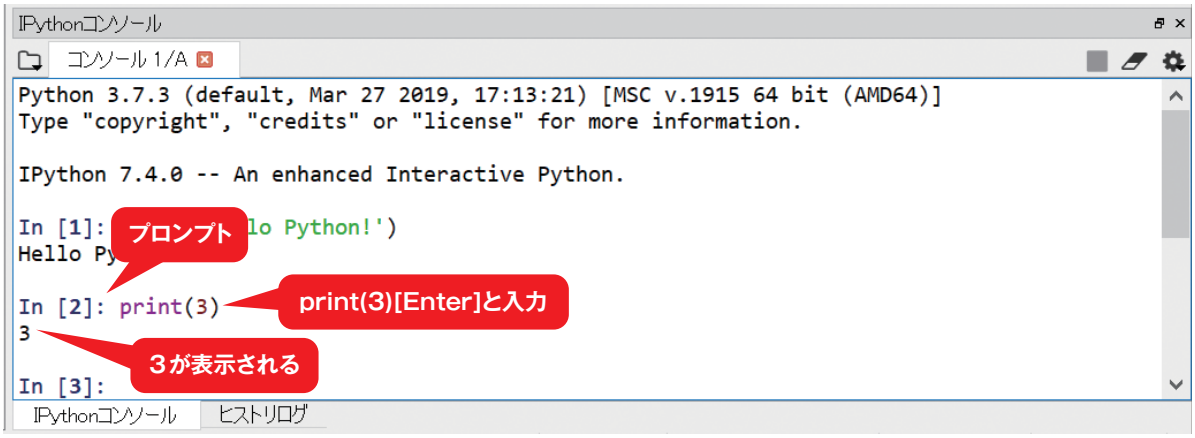
図5●SpyderのIPythonコンソールにコードを入力



1日目 Part 3 数値と文字列

講師 Pythonは、文字列を表示することができますが、数値を文字列で表示することもできます。例えば、数値3をprint関数で表示してみましょう（**図1**）。

図1●数値の表示



進 ん? 文字列と何が違うんや?

講師 Pythonでは、「'」（シングルクォーテーション）か「"」（ダブルクォーテーション）で括ると「文字列」、括らないと「数値や変数」という文字列以外の値となります。数値を記述すると、計算もできます。「print(3 + 4)」[Enter]と入力すると、7と表示されます。

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915
64 bit (AMD64)]
Type "copyright", "credits" or "license" for more informat
ion.

IPython 7.4.0 -- An enhanced Interactive Python.


In [1]: print('Hello Python!')
Hello Python!

In [2]: print(3)
3

In [3]: print(3 + 4)
7
```

print(3 + 4)[Enter]と入力

計算結果が表示される

講師 では、「3 + 4」をシングルクォーテーションで括って、「print('3 + 4')」にするとどうなるでしょう?

シングルクォーテーションで括る

```
In [4]: print('3 + 4')
3 + 4
```

式が文字列として表示される

愛 文字列に…なぜ。

講師 Pythonでは、たとえ数字や計算式でも、シングルクォーテーションで括ると「文字列」になります。そのため、'3'や'A'や'あ'は「文字列」です。

進 数字も、シングルクォーテーションで括ると文字列かぁ。

講師 ここで、文字列同士を「+」記号で足してみましょう。すると「文字列連結」の処理になります。例えば、「print('3' + '4')」を実行すると「34」と表示されます。

文字列3と文字列4の連結

```
In [5]: print('3' + '4')
34
```

連結されて表示

進 ホントだ。これはおもろいのう。

講師 文字列の連結演算子で注意すべき点は、数値と文字列の連結です。「print(3 + '4')」のようなコードはエラーになります。

数値3と文字列4の足し算

```
In [6]: print(3 + '4')
Traceback (most recent call last):

  File "<ipython-input-7-15e3e7e599e4>", line 1, in <module>
    print(3 + '4')

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

3は数値、'4'は文字列なので、加算も連結もできずTypeErrorになる

講師 このような場合は、int関数やfloat関数を使って文字列を数値に変換するか、str関数を使って数値を文字列に変換してから処理します。

str関数で数値の3を文字列'3'に変換

```
In [6]: print(str(3) + '4')
34
```

文字列同士の連結

```
In [7]: print(3 + int('4'))
7
```

数値同士の足し算

int関数を使って文字列'4'を、数値の4に変換

愛 変換には、関数を…。

講師 今の段階では、文字列と数値は関数で変換が可能であることがわかれば大丈夫。

関数の使い方はすぐ慣れますよ。ところで、Pythonで掛け算は「*」（アスタリスク）記号を使います。この記号を、数値に使えば掛け算なわけですが、文字列に使うと文字列の繰り返し表示になります。

通常のかけ算

```
In [8]: print(3 * 4)
12

In [9]: print('3' * 4)
3333
```

文字列の繰り返し

3を4回繰り返した

愛 アスタリスク、繰り返し…。

講師 このように、Pythonのコードに登場する「値」には、文字列や数値といったデータの種類があります。このデータの種類のことを「データ型」と呼びます。よく利用するデータ型には、数値型、文字列型、それに「真偽値型」なんかもあります（表1）。

表1●データ型

データ型	例
数値型	0　10　3.14　-3 など
文字列型	'Hello'　" あいうえお " など
真偽値型	True または False

進 True、Falseって何?

講師 この値は、条件を判断するときに用いる特殊な値で、Trueは「成り立つ」、Falseは「成り立たない」という意味があります。この値も、今は「こんな値もあるんだ」って感じで大丈夫です。

進 なるほど。

愛 …。

講師 最後にエスケープシーケンスについて説明します。文字列の中に、エスケープ文字を入れることで、改行したりタブスペースを入れたりできます。例えば、文字列の中に改行を入れたい場合は、「¥n」というエスケープシーケンスを入れます。

```
In[10]: print('ABC¥nDEF')
ABC
DEF
```

¥nの場所で改行して表示される

自動的に改行して表示される

講師 エスケープシーケンスは、文字列を表現するためのシングルクォーテーションやダブルクォーテーションを、文字として表示したいときや、¥記号を文字として表示したいときなどに利用します。エスケープシーケンスで、よく利用するものを**表2**にまとめました。

表2●エスケープシーケンス

文字	意味
¥newline	¥記号の後ろの改行を無視
¥¥	¥記号の表示
¥'	シングルクオートの表示
¥''	ダブルクオートの表示
¥n	改行
¥r	復帰

※Windows環境の¥記号は、macOSやLinux環境ではバックスラッシュ記号（\）になります。

進 ¥newlineって何?

講師 プログラムのコードを任意の場所で改行したいとき使います。例えば、print('ABC

DEFG')というコードの文字列の途中で[Enter]で改行して、残りを入力後、[Enter]キーで実行すると、エラーになります。

```
In[11]: print('ABC
   ...: DEF')
  File "<ipython-input-10-a82b40021a12>", line 1
    print('ABC
                ^
SyntaxError: EOL while scanning string literal
```

print('ABC[Enter]

DEF')[Enter]

SyntaxErrorになる

講師 ところが、¥記号を入れてから改行すると、その改行はなかったものとして「継続入力」の意味になり、エラーになりません。

```
In[12]: print('ABC¥
   ...: DEF')
ABCDEF
```

print('ABC¥[Enter]

DEF')[Enter]

継続されるので、エラーにならない

※Spyderのコンソールやエディターでは、¥記号は＼で表示されます。

愛 使い道は?

講師 とても長い1行のコードを、見た目上、改行する場合などに利用します。それでは、今日はここまでにしましょう。

2日目

変数と演算

Part 1 変数

講師 おはようございます。今日は、「変数と演算」についてのお話です。プログラミングを学習するとき、最初に理解する必要がある概念として「変数」があります。変数は、「値を入れる入れ物」と説明されることが多いのですが、私としては「値に貼り付けるラベル」の例えで説明したいですね（**図1**）。こちらの方が、今後「オブジェクト」の概念が登場したときに、正しい理解につながると思います。

図1●変数のイメージ

進 値にラベルを貼り付けて、どうすんの?

講師 例えば、ユーザーが入力した値を使って計算したい場合、計算式はプログラムできますが、値は入力するまでわかりませんね。そこで、入力値には変数というラベルが貼り付けられると仮定します。すると、ラベル、つまり変数を使って先に計算式のプログラムを作ることができます。

進 なるほど! でも、その説明だと、変数は「値を入れる入れ物」の例えの方がしっくりくるな。

講師 今の段階だと、そうかもしれませんね。ただ、最初のイメージが肝心です。変数は「値に貼り付けるラベル」のイメージで理解するようにしてください。

愛 …了解。

講師 先ほどの図1のイメージをPythonのコードで表現すると、「x = 10」のようになります。「=」（イコール）記号は「代入演算子」と言います。数学で出てくる「=」は「等しい」の意味ですが、Pythonではそうではない点に注意してください。

進 紛らわしいな〜。

講師 もっと紛らわしいことを言うと、「=」は日本語では「代入演算子」と呼ばれることが多いのですが、英語では「assignment operator」です。「assignment operator」を直訳すると「割り当て演算子」なのですね。「x = 10」も、「xに10を代入する」ではなく、「10にxを割り当てる」と考えた方が良いと思います。先ほどの例えを使うと、「10にxというラベルを割り当てる」です。

進 ???

講師 ま、まあ、ちょっと細か過ぎることを言ってしまいましたね。今の話は頭の片隅にでも入れておいてもらえば十分です。

進 りょ、了解です!

講師 この変数xの値は、print関数で表示することで確認できます。また、SpyderのIPythonコンソールには、変数などの値を直接表示する機能があるので、こちらも試しましょう。

愛 Spyder、起動!

講師 IPythonコンソールが表示されたら（**図2**）、次のコードを入力して確認しましょう。

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915
64 bit (AMD64)]
Type "copyright", "credits" or "license" for more informat
ion.

IPython 7.4.0 -- An enhanced Interactive Python.

In [1]: x = 10

In [2]: print(x)
10

In [3]: x
Out[3]: 10
```

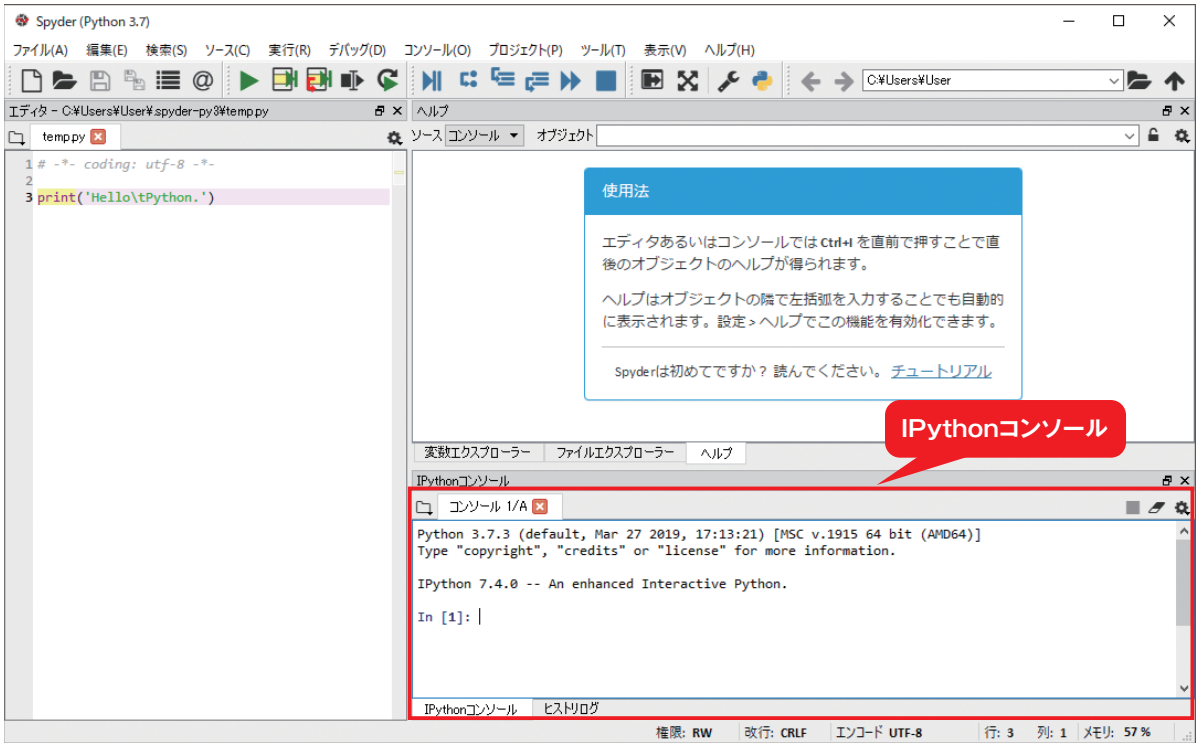
10に変数xを割り当てて、変数xを利用できるようにする

変数xの値をprint関数で確認

x[Enter]でも、変数xの値を確認できる

進 値に変数を「割り当て」るんやな。

図2●IPythonコンソールの利用



講師 今回は、「x」という変数名を付けましたが、変数名には、次の決まりがあります。

(1) 1文字目は、半角英文字かアンダースコア(_)
(2) 2文字目以降は、半角英数文字かアンダースコア
(3)キーワード(予約語)は使えない

講師 一般的に、意味のある変数の場合は、その意味を表す小文字の英単語を、変数名に利用します。もし、複数の単語を使いたい時は、アンダースコアで区切ります。このとき、数字を1文字目に使うことはできません。例えば、年齢を表す変数なら「age」、横幅widthと高さheightを1つの変数で表すなら「width_height」のようにします。

```
In [4]:age = 24

In [5]:print('年齢は、'+ str(age) + '才です。')
年齢は、24才です。

In [6]:width_height = 100

In [7]:print('幅と高さは' + str(width_height) + 'cm。')
幅と高さは100cm。
```

変数ageの値を24で初期化

変数の値は24なので、str関数で文字列に変換して連結する

複数の単語を変数名にする場合はアンダースコアでつなぐ

講師 また、Pythonには文法的な意味を持つ単語があり、これを「キーワード」とか「予約語」と呼びます(**表1**)。

講師 キーワードと同じ単語は、変数名には使えません。したがって、classという変数は作れないので、例えばclass_1のような変数名にします。

講師 ちなみに、変数名などの決め方を「命名規則」と呼びます。「width_height」のように、単語をアンダースコアで接続して書くと、変数名がヘビのように見えませんか?

表1●キーワード。予約語とも呼ばれる

False	class	finally	is
return	None	continue	for
lambda	try	True	def
from	nonlocal	while	and
del	global	not	with
as	elif	if	or
yield	assert	else	import
pass	break	except	in
raise	async	await	

classはPythonのキーワードなので、変数名に使えない

```
In [8]:  class = 'マダイ'
 File "<ipython-input-8-68daffa0ad53>", line 1
    class = 'マダイ'
          ^
SyntaxError: invalid syntax


In [9]:  class_1 = 'マダイ'

In [10]: class_1
Out[10]: 'マダイ'
```

class_1なら変数名として使うことができる

愛 ヘビ…スネーク…好き…。

講師 そうです。スネーク。そして、このような命名規則を「スネークケース」と呼んだりします。スネークケースは、Python以外のプログラミング言語やデータベースでもよく使われるので覚えておくと良いでしょう。

Part 2 2日目 演算

講師 それでは、Pythonの演算子を紹介していきます。演算子は、変数などの値を演算したり、値を変更したりするときに利用する、記号やキーワードです。演算子が作用する変数や値のことを「オペランド」と呼びます。

愛 オペランド…演算子が作用する値…。

講師 …。まず最初に、算術演算子を紹介します。Pythonの算術演算子には、**表1**の種類があります。

講師 Pythonの算術演算子では、掛け算と割り算が算数の記号とは違っているので注意してください。また、除算演算子が2種類あります。スラッシュが1つの除算演算子は演算結果が小数付きになります。スラッシュが2つの方は小数点以下を切り捨てます。

表1●算術演算子

演算子	意味	例
+	加算	3 + 4
−	減算	6 − 2
*	乗算	2 * 4
/	除算	8 / 2
//	除算	8 // 2
%	剰余	5 % 2
**	べき乗	2 ** 3

```
In [1]: 15 / 4
Out[1]: 3.75
```
/ だけだと小数点以下も表示される

```
In [2]: 15 // 4
Out[2]: 3
```
// だと、小数点以下は切り捨てられる

講師 剰余演算子は、割った余りを求めます。べき乗は、後ろのオペランド分だけ繰り返し掛け算します。

```
In [3]: 15 % 4
Out[3]: 3
```
割り算の余りが表示される

```
In [4]: 2 ** 4
Out[4]: 16
```
2×2×2×2で16

講師 そのほかには、データをビット単位で演算するビット演算子もあります（**表2**）。IoT系の制御プログラムを作るときに重宝する演算子です。

表2●ビット演算子

演算子	意味	例
&	論理積（AND）	0b10 & 0b01 → 0
|	論理和（OR）	0b10 | 0b01 → 3
^	排他的論理和（XOR）	0b10 ^ 0b11 → 1
<<	ビットシフト（左シフト）	0b0001 << 2 → 4
>>	ビットシフト（右シフト）	0b1000 >> 2 → 2
~	ビット反転（NOT）	~1 → -2

進 ビット演算子の「AND」って、何じゃ?

講師 ANDは論理和ですから、双方のオペランドのビットが1なら1、それ以外は0になり

ます。0bから始まる数字は2進数を表します。したがって、10と01の場合、00になるので10進数の0になります。

愛 2進数の「10」と「01」のORは、2進数の11つまり3。11と10のXORは、双方1なら0だから01、つまり1。

進 愛ちゃん、すごいな。なるほど。シフトはビットをずらすってことじゃな。じゃけど、ビット反転はどうしてマイナスになるんじゃ?

講師 コンピュータでは、「2の補数」を使って負の値を表現します。その結果、データの最上位ビットが1のとき負になります。

進 つまり、データの0の部分がすべて1に反転したから、マイナスになったんかい。

講師 そうです。ビット演算については、今回はここまでにしておきましょう。

愛 ザンネン…。

講師 次は、代入演算子です。先ほど変数の説明のところでも言いましたが、「割り当て演算子」という方が適切だと思いますが、一般には代入演算子と呼ばれます(**表3**)。この演算子は、値に変数を割り当てるときに使います。単純な割り当て以外に、計算結果を再度割り当てる演算子があります。うまく使うとコードが短くなり、見通しが良くなります。

講師 一通り、試してみましょう。まずは、単純に1に変数aを割り当ててみます。このように、初めて変数が値に割り当てられることを「変数の初期化」と呼びます。

```
In [5]: a = 1        ← 1に変数aを割り当てる(変数の初期化)

In [6]: a            ← 変数aの値を確認する
Out[6]: 1
```

表3●割り当て演算子（代入演算子）

演算子	説明	同じ式
=	単純割り当て（値に変数を割り当てる）	a = 3は、3に変数a を割り当て
+=	加算割り当て（変数に右側の値を加えた値に、再度変数を割り当てる）	a += b は、a = a + b と同じ
−=	減算割り当て（変数から右側の値を引いた値に、再度変数を割り当てる）	a −= b は、a = a − b と同じ
*=	乗算割り当て（変数に右側の値を掛けた値に、再度変数を割り当てる）	a *= b は、a = a * b と同じ
/=	除算割り当て（変数を右側の値で割った値に、再度変数を割り当てる）	a /= b は、a = a / b と同じ
//=	除算割り当て（変数を右側の値で割った値に、再度変数を割り当てる）	a //= b は、a = a // b と同じ
**=	べき乗割り当て（変数を右側の値でべき乗した値に、再度変数を割り当てる）	a **= b は、a = a ** b と同じ
%=	剰余割り当て（変数を右側の値で割った余りに、再度変数を割り当てる）	a %= b は、a = a % b と同じ

講師 そのほかの割り当て演算子は、加算や減算などとの組み合わせになります。試しに、「+=」を使ってみましょう。

```
In [7]: a += 10

In [8]: a
Out[8]: 11
```

a = a + 10と同じ

変数aの値を確認する

愛 カンタン…。

講師 割り当て演算子は以上です。それでは、比較演算子に進みましょう。

Part 3 比較演算子と論理演算子

講師 次に紹介する演算子は、「比較演算子」です。比較演算子は、両端の「オペランド」を比較して、「True」か「False」を返します。例えば、両端のオペランドが等しいかどうかを調べる比較演算子は、「==」のようにイコールを2つ連続して記述します。もし、数値の10と10が等しいかどうかを調べるなら、10 == 10のように書きます。

10と10が等しいか比較演算子で調べる

```
In [1]: 10 == 10
Out[1]: True
```

成り立つ場合はTrueという値になる

愛 等しいとき…True…。

進 10が10と等しいって、当たり前すぎるな。

講師 あくまでも説明のための例なので…。比較演算子は、「成り立つ」と判断した場合はTrueになりますが、成り立たない場合はFalseになります。

10と20が等しいか調べる

```
In [2]: 10 == 20
Out[2]: False
```

成り立たないのでFalseになる

講師 もちろん、変数の値を比較することもできます。例えば、変数xとyが「10」で、変数strが「'10'」だとします。この場合、xとyは数値で、strは文字列なので、xとyは等しいですが、strと比較するとFalseになります。

```
In [3]: x = 10
```
変数xを10で初期化

```
In [4]: y = 10
```
変数yを10で初期化

```
In [5]: str = '10'
```
変数strを'10'で初期化

```
In [6]: x == y
Out[6]: True
```
xとyはともに10。等しいのでTrue

```
In [7]: x == str
Out[7]: False
```
xは10、strは'10'。等しくないのでFalse

講師 このような比較演算子には、==以外にも**表1**のような種類があります。

表1●比較演算子

演算子	意味
==	a == b において、b と a が等しいときに True
!=	a != b において、b と a が等しくないときに True
>	a > b において、b より a が大きいときに True
<	a < b において、b より a が小さいときに True
>=	a >= b において、b と a が等しいか、b より a が大きいときに True
<=	a <= b において、b と a が等しいか、b より a が小さいときに True

進 比較は=が2つ、割り当ては1つか～い。本当に紛らわしいのう。

講師 ちなみに、比較演算子が返す値はTrueかFalseなので、真偽値型になります。この真偽値型は、数字の1と0と互換性があります。したがって、次のような比較も、可能

です。

```
In [8]: True == 1
Out[8]: True
```
Trueは1と等しい

```
In [9]: True == 0
Out[9]: False
```
Trueは0と等しくない

```
In [10]:False == 1
Out[10]:False
```
Falseは1と等しくない

```
In [11]:False == 0
Out[11]:True
```
Falseは0と等しい

講師 最後は、「論理演算子」です。論理演算子は、両端のオペランドがTrueかFalseかを比較する演算子です。通常は、比較演算子と組み合わせて、複雑な比較を行うときに使用します。論理演算子には、**表2**の種類があります。

表2●論理演算子

演算子	意味
and	左から評価し、Falseと同等になった項があればその時点でその項の値を返し、すべてTrueと同等であれば最後の項の値を返す
or	左から評価し、Trueと同等になった項があればその時点でその項の値を返し、すべてFalseと同等であれば最後の項の値を返す
not	not aにおいて、aがTrueならばFalse、FalseならばTrue

講師 and演算子とor演算子は、結果をTrueかFalseで返すのではなく、評価が決まった時点でその項の値を返します。その項の値がTrueかFalseになることはあります。実際に確認してみましょう。

```
In [12]: 3 < 5 and 6 < 8
Out[12]: True
```
and演算子はどちらのオペランドもTrueならばTrue

```
In [13]: 3 < 5 or 6 < 8
Out[13]: True
```
or演算子はどちらかのオペランドがTrueならばTrue

```
In [14]: 1 and 2
Out[14]: 2
```
1はTrueと等しいので、次に2が評価され、その2が返る

```
In [15]: 1 or 0
Out[15]: 1
```
1はTrueと等しいので評価が決定し、その1が返る

進 論理演算子のオペランドに数値を使うと、ややこしくなるのう。

講師 通常、論理演算子のオペランドには真偽値型、つまりTrueかFalseがきます。ただ、数字や文字列を使ってもエラーにはならないので注意してください。

愛 フッ…。

進 愛ちゃん、なぜ不敵な笑いを…。

講師 …。Pythonの演算子には、割り当て演算子、算術演算子、比較演算子、論理演算子以外にもいくつかあります。それぞれの演算子には、優先順位があるので確認しましょう（**図1**）。

進 まだまだ、知らん演算子がたくさんあるのぉ。

講師 これらの演算子の優先順位は、無理に全部覚える必要はありません。括弧内の演算が優先されるのは算数と同じなので、忘れたときは括弧で演算順序を決める方が確実で見やすくなると思います。

図1●演算子の優先順位

優先順位	演算子
高	await
↑	**
↑	+ − ~ ※値や変数につける（例. −a）
↑	* @ / % //
↑	+ −
↑	<< >>
↑	&
↑	^
↑	|
↑	< <= > >= == != < > is is not in not in
↑	not
↑	and
↑	or
低	lambda

```
In [16]: x = 10
```

論理演算子より比較演算子の方が優先順が高い

```
In [17]: x >= 0 and x <= 100
Out[17]: True
```

変数xが0以上、100以下なのでTrue

```
In [18]: (x >= 0) and (x <= 100)
Out[18]: True
```

優先順は変わらないが、こちらの方が見やすい場合がある

講師 ちなみに、変数の値がある範囲内にあるかどうかを調べたいときは、Pythonでは次のように比較演算子の組み合わせだけでも記述できます。

```
In [19]: x = 10

In [19]: 0 <= x <=100
Out[19]: True
```

論理演算子を使わない記述

変数xが0以上、100以下なのでTrue

愛 キレイなコード…。

講師 今日は、ここまでにしましょう。明日は、制御文のお話をします。

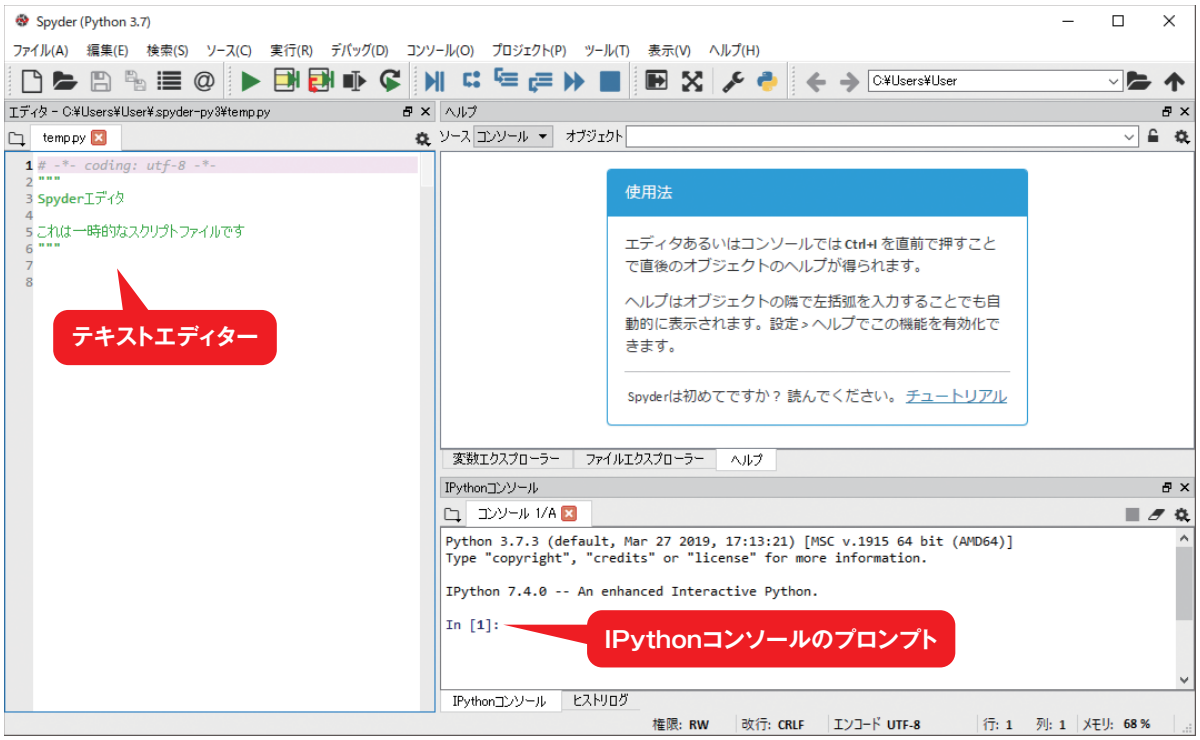
3日目

制御文とデータ構造

Part 1 制御文

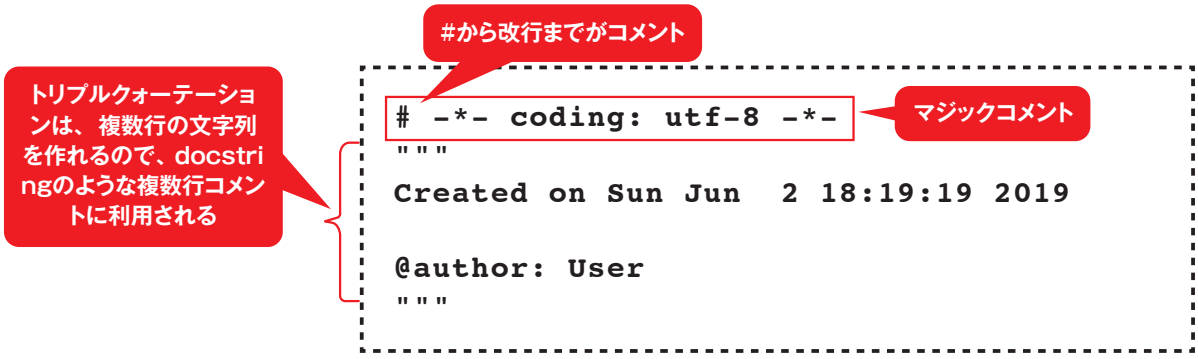
講師 研修3日目の今日は、「制御文」と「データ構造」についてお話しますが、その前に、エディターを使ってプログラムを入力し、実行する方法を説明します。

図1●起動直後のSpyder



進 なんじゃ、IPythonコンソールは使かわんの?

講師 いえ、実行はIPythonコンソールで行います。Spyderの左側の領域を見てください。ここが、テキストエディターになっています。すでに、次のコードが自動入力されていると思います。



講師 最初の「# -*- coding: utf-8 -*-」の部分は「マジックコメント」と呼ばれています。このように、半角の「#」から改行までが、注釈、つまりコメントとして扱われ、プログラムの処理には影響を与えません。

進 でも、なぜ「マジック」なんじゃろ。

愛 ソースコードの文字エンコーディングを指定…。

講師 その通り。Pythonのバージョン2.xでは、ソースコードの文字はデフォルトがASCIIだったので、日本語などが文字化けしないように、ソースコードに「マジックコメント」を記述して文字エンコーディングを指定していました。もっとも、Python3.xのデフォルトは「UTF-8」なので、今回「マジックコメント」は必要ありません。

進 その下は、何じゃ?

講師 シングルクォーテーションかダブルクォテーションを3つつなげた「トリプルクォーテーション」('''または""")で文字列を囲むと、改行を含めた文字列を生成することができます。単独の文字列をソースコード中に記述しても、処理には影響が出ません。そのような文字列はコメントとして利用できます。

愛 docstring(ドクストリング)にも…。

講師 そ、その通りですね。関数やクラスの定義をするとき、その関数やクラスがどのようなものか、作者は誰か、作成日時はいつかなどのコメントを入れることをdocstringと呼びます。トリプルクォーテーションは、このdocstringにも利用されます。

進 だから、日付やauthor（著者）の記述があるんじゃな。

講師 こちらも、今は使わないので削除してかまいません。それでは、エディターに、次のprint関数を入力してください。

エディターにprint関数を3行入力

```
print('中島 省吾')
print('小鯖 進')
print('栄井 愛')
```

講師 エディターに入力できたら、実行します。実行は、Spyderの上、中央付近にある「ファイルを実行」ボタンをクリックします。すると、ソースファイルを保存する場所と名前を指定するウインドウが開くので、今回はCドライブのルートにPythonというフォルダーを作ってsample01.pyというファイル名で保存します。ちなみに、拡張子は通常「.py」を使います。保存が完了すると、IPythonで実行されます（**図2**）。

進 コンソールに、名前が出たのじゃ。

講師 print関数の順番に表示されていますね。このような処理の流れを、「逐次」とか「順次」と呼びます。つまり、プログラムは基本的に上から順番に処理が実行されていきます。

進 そりゃそうじゃろ。

講師 ただ、このままでは、単純な処理しか実行できません。そこで、条件を判断して処理の流れを変える機能が用意されています。これを、「分岐」とか「選択」と呼びます。Pythonでは、「if文」を使って分岐を記述します。sample02.pyのコードをエディターに入力して実行してみましょう（**図3**）。

図2●ソースファイルの実行

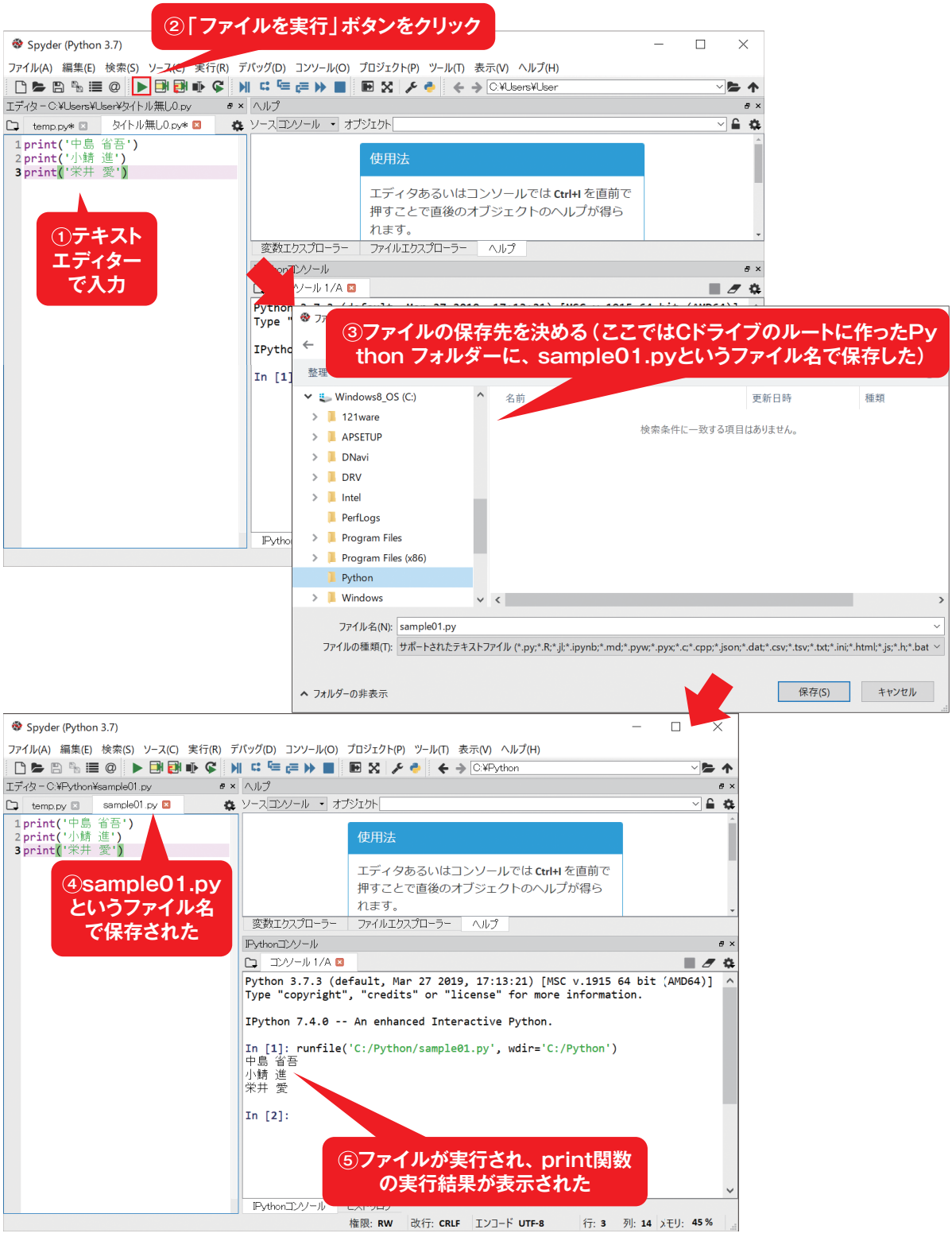
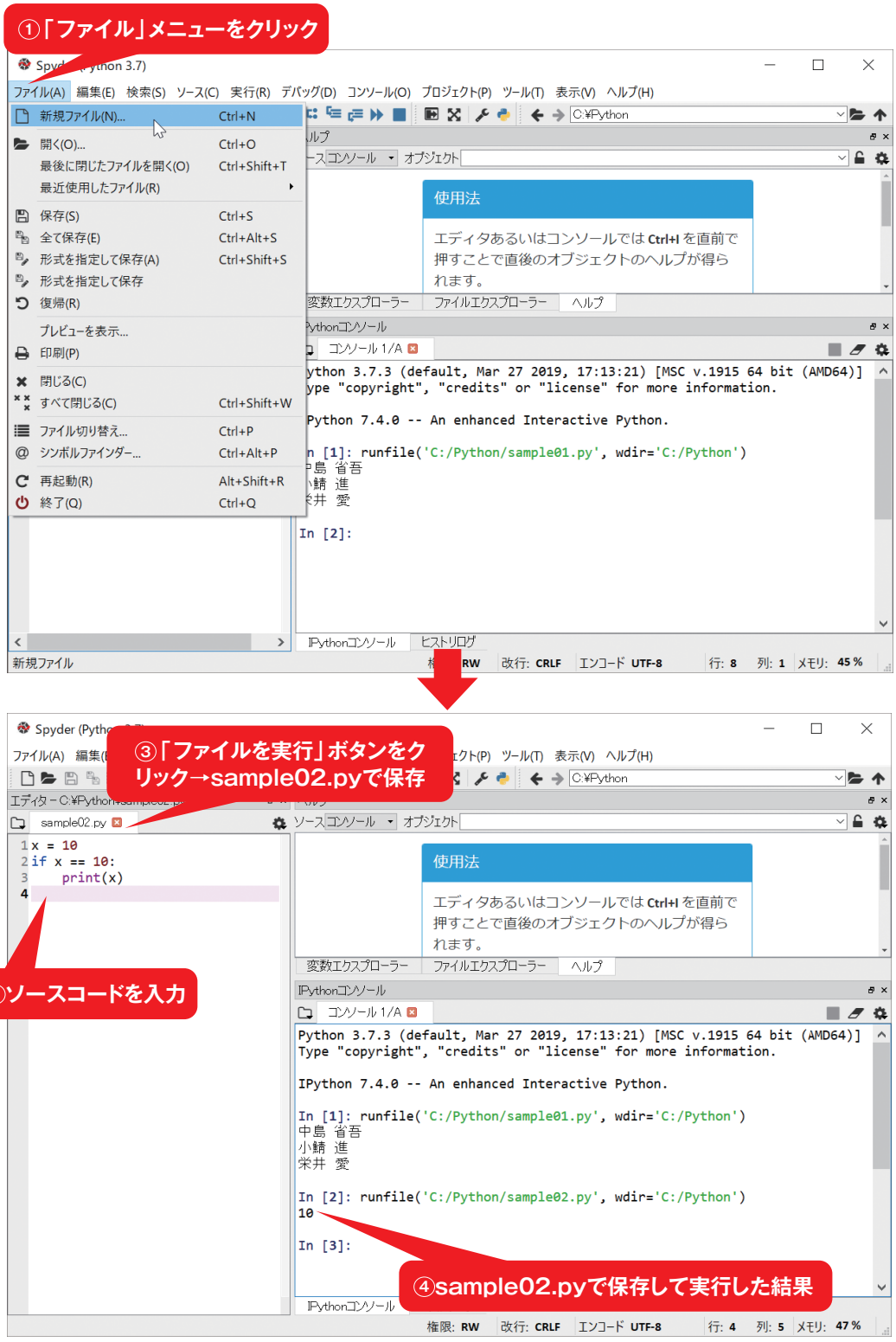


図3●sample02.py（次ページ）を実行



sample02.py

```
x = 10

if x == 10:
    print(x)
```

:(コロン)の後ろで改行

インデント(字下げ)を行う

講師 ifの後ろでは、変数xが10と等しいかどうかを比較演算子で調べています。これが「条件式」になります。最初の行でxは10に初期化されているので、比較演算子はTrueを返します。if文は条件式がTrueの場合、ブロック内の処理を実行します。その結果、図3のコンソールのように10が表示されます。

愛 ブロック!…ブロック!

進 なんや、愛ちゃんが喜んでます。

講師 「:」(コロン)で改行すると、次の行からif文のブロックが開始されます。if文のブロックは、一般的には、半角スペースを4つ入れて「インデント」(字下げ)を作り、ブロックとします。もし、同じブロックの中に複数のコードを書きたいときは、同じインデントの幅にして記述できます。

sample02.py

```
x = 10

if x == 10:
    print('ここは、if文のブロックです。')
    print(x)
```

同じインデント(字下げ)の幅なら同じブロックになる

IPythonコンソール

```
In [3]: runfile('C:/Python/sample02.py', wdir='C:/Python')
ここは、if文のブロックです。
10

In [4]:
```

条件式がTrueなら、if文のブロックが実行される

講師 では、xの値を1に変更して、ブロック内の処理が実行されないことも確認してみましょう。

sample02.py

```
x = 1

if x == 10:
    print('ここは、if文のブロックです。')
    print(x)
```

10を1に変更

IPythonコンソール

```
In [4]: runfile('C:/Python/sample02.py', wdir='C:/Python')

In [5]:
```

条件式がFalseだと、if文のブロックは実行されない

講師 このコードでは「x = 1」と記述して変数の値を固定していますから、条件式の判断結果は常に同じです。そこで、input関数を使って入力された値で判断するようにしてみます。input関数は、キーボードからの入力を文字列で返します。この入力文字列に変数を割り当てて、文字列「'10'」と比較してみましょう。

input関数は、キーボードからの入力を文字列で返す

sample02.py

```
x = input('10を入力してください:')

if x == '10':
    print('正解')
```

入力された文字列が'10'と同じか調べる

IPythonコンソール

```
In [5]: runfile('C:/Python/sample02.py', wdir='C:/Python')

10を入力してください:10
正解

In [6]:
```

10を入力して[Enter]キーを押す

if文のブロックが実行された

講師 キーボードから入力された文字列を数値に変換するときは、整数ならint関数を使います。

sample02.py

```
x = input('10を入力してください:')
x = int(x)

if x == 10:
    print('正解')
```

文字列を整数に変換

10と同じか調べる

IPythonコンソール

```
In [5]: runfile('C:/Python/sample02.py', wdir='C:/Python')

10を入力してください:10
正解

In [6]:
```

10を入力して[Enter]キーを押す

if文のブロックが実行された

講師 このままでは、条件式がTrueならブロックの中を実行しますが、Falseのときは何もしません。もしFalseの場合に処理を実行したいときは、else文を使います。

sample02.py

```
x = input('10を入力してください:')
x = int(x)

if x == 10:
    print('正解')
else:
    print('不正解')
```

文字列を整数に変換

else文のブロック

IPythonコンソール

```
In [5]: runfile('C:/Python/sample02.py', wdir='C:/Python')

10を入力してください:6
不正解

In [6]:
```

10以外の数値を入力して[Enter]キーを押す

else文のブロックが実行された

講師 また、何もしないブロック作りたいときは、passキーワードを使います。

sample02.py

```
x = input('10を入力してください:')
x = int(x)

if x == 10:
    pass
else:
    print('不正解')
```

文字列を整数に変換

何もしない場合でも、ブロックを作るためにpassキーワードが必要

IPythonコンソール

```
In [5]: runfile('C:/Python/sample02.py', wdir='C:/Python')
10を入力してください：10
In [6]:
```

10を入力して[Enter]キーを押す

何も実行しない

講師 Pythonでは、インデントを使ってブロックを作るので、仮に何もしない場合でも、passキーワードを使ってインデントを作ります。次は、繰り返し（ループ）のお話です。

Part 2 3日目 反復処理

講師 では、反復処理の話です。反復処理とはブロック内の処理を繰り返すことです。「繰り返し」または「ループ」とも呼ばれます。反復処理の実現方法はいくつかありますが、ここでは「while文」を説明しましょう。記述の仕方は、whileと書いて半角スペースを置き、その後ろに条件式を書きます。そして、コロン、改行ときて、条件式がTrueの場合に繰り返すブロックを書きます。「i=i+1」は、数学的にはギョッとする式ですが、「=」は単純割り当ての演算子ですから、「iとi+1は等しい」という意味ではありません。これは、「変数iに1を加えた値に、再度変数iを割り当てる」という意味です。つまり、変数iの値を+1する処理になります。

sample03.py

```
i = 0

while i < 3:
    print('さかな')
    i = i + 1
```

繰り返すかどうかの条件式

while文のブロック

インデント

IPythonコンソール

```
In [1]: runfile('C:/Python/sample03.py', wdir='C:/Python')
さかな
さかな
さかな

In [2]:
```

条件式がTrueの間、while文のブロックが繰り返される

講師 もし、無限ループになって止まらなくなった場合は、IPythonコンソール内をクリックしてから、[Ctrl]キーを押しながら[C]キーを押して強制終了します。

sample03.py

```
i = 0

while i < 3:
    print('さかな')
    # i = i + 1
```

コメントにして、iの値が増えないようにする

IPythonコンソール

```
In [2]: runfile('C:/Python/sample03.py', wdir='C:/Python')
さかな
さかな
さかな
さかな
さかな
さかな
さかな
・・・
```

無限ループになってしまった場合は、IPythonコンソール内をクリックしてから、[Ctrl]キーを押しながら[C]キーを押して強制終了する

講師 反復処理の制御を強制的に変更するキーワードに、「break文」と「continue文」があります。break文は、強制的に繰り返しのブロックから抜けます。

sample03.py

```
i = 1

while i < 5:
    if i == 4:
        break
    print(str(i) + ':さかな')
    i = i + 1
```

変数iが4と等しいときはif文のブロックから抜ける

ブロックから抜けて次の処理へ

IPythonコンソール

```
In [3]: runfile('C:/Python/sample03.py', wdir='C:/Python')
1:さかな
2:さかな
3:さかな

In [4]:
```

iが4になると、if文のブロックから抜けてしまうため、プログラムが終了した

講師 continue文は実行されると、ループの先頭に戻ります。

sample03.py

```
i = 0

while i < 5:
    i = i + 1
    if i == 2:
        continue
    print(str(i) + ':さかな')
```

変数iが2と等しいときはループの最初に戻る

IPythonコンソール

```
In [4]: runfile('C:/Python/sample03.py', wdir='C:/Python')
1:さかな
3:さかな
4:さかな
5:さかな

In [5]:
```

変数iが2のときだけ、print関数が実行されていない

3日目 Part 3 リスト

講師 それでは、今日の最後に「リスト」と「辞書」を紹介します。

進 リストって一覧表じゃろ?

講師 Pythonの「リスト」は、表ではありません。複数のデータをまとめて保持する「オブジェクト」です。

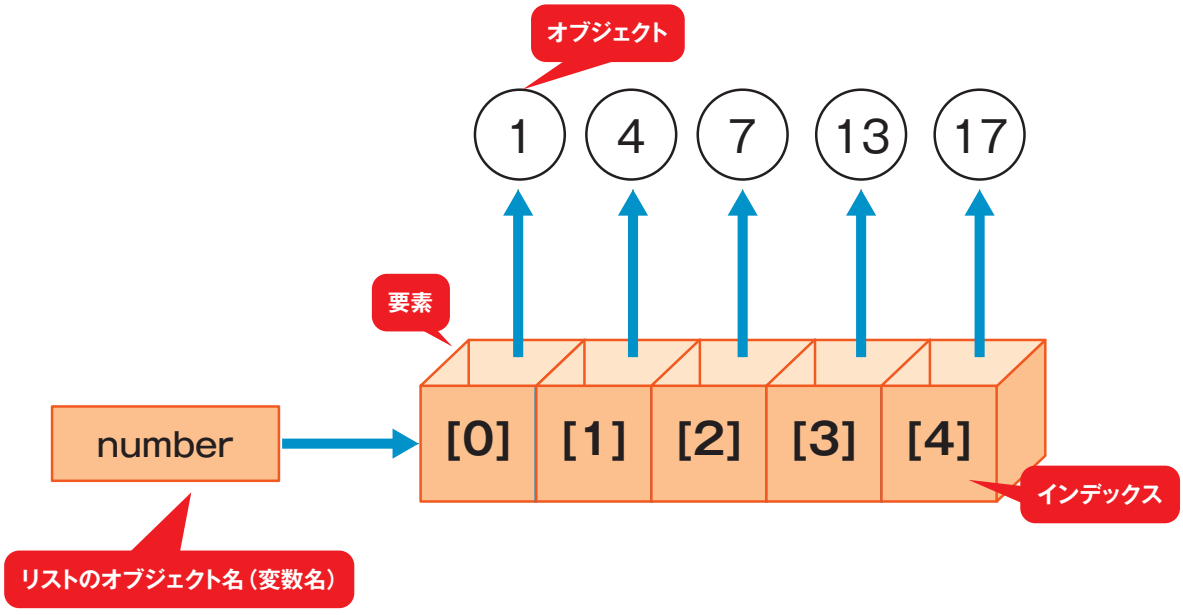
愛 私も…オブジェクト…。

講師 実はPythonでは、値や変数、文字、文字列、この後に紹介する関数など、プログラムを構成する様々な要素を「オブジェクト」として扱います。そして、このような考え方をするプログラミング言語を「オブジェクト指向言語」と呼びます。

進 つまり、リストも辞書も何もかもオブジェクトなんじゃな。

講師 そうです。リストは、内部に複数のオブジェクトを保持することができます。そのオブジェクトを「要素」と呼びます。個々の要素には「インデックス」という番号が振られます。この番号を使って、リスト内のオブジェクトを利用します(**図1**)。

図1●リストのオブジェクトのイメージ



進 名前より、番号の方が管理しやすいからじゃろう。

講師 ただ、名前でアクセスするオブジェクトもあります。

進 なに?

講師 それが「辞書」というデータ構造です。辞書については後で話しますので、まずはリストの作り方から説明しましょう。リストは次のように、角括弧の中にリストに格納したいオブジェクトをカンマ区切りで記述し、変数に割り当てます。これで、複数のデータを内部に保持するリストができあがります。リスト内の各データを取得したり表示したりするには、変数名に角括弧を付けて、インデックスを指定します。

sample04.py

```
number = [1, 4, 7]

print(number[0])
print(number[1])
print(number[2])
print(number[-2])
```

3つの要素を持ったリストオブジェクトの生成

リストの要素を表示（インデックスは0から始まる）

インデックスが−2の場合は、後ろから数えて2番目の要素を表示

IPythonコンソール

```
In [1]: runfile('C:/Python/sample04.py', wdir='C:/Python')
1
4
7
4

In [2]:
```

リストの要素が持つ値

後ろから2番目の要素の値

講師 もし、リストの中に特定のデータがあるかどうかを調べたいときは、if文の条件式で「in演算子」や「not in演算子」を使います。

sample04.py

```
number = [1, 4, 7, 13, 17]

if 4 in number:
    print("numberには、4が含まれます。")

if 5 not in number:
    print("numberには、5が含まれまれません。")
```

in演算子は、オブジェクト内にデータがあればTrueを返す

not in演算子は、オブジェクト内にデータがなければTrueを返す

IPythonコンソール

```
In [3]: runfile('C:/Python/sample04.py', wdir='C:/Python')
numberには、4が含まれます。
numberには、5が含まれまれません。

In [4]:
```

リスト内の要素に4が含まれている

リスト内の要素に5が含まれていない

講師 また、リストには「スライス」という、少し複雑ですが、便利な機能があります。例えば、5個の要素を持つリストから、インデックスの1から3の部分を取り出したいときなどに使います。スライスはリストの角括弧の中に、「開始インデックス：終了インデックス」のように記述します。

sample04.py

```
number = [1, 4, 7, 13, 17]

print(number[1: 4])

print(number[: 3])

print(number[1: ])

print(number[1: 4: 2])
```

インデックス1～3まで。終了インデックスは含まれない

先頭からインデックス2まで

インデックス1から最後まで

インデックス1～3まで（最後の数字はステップ数）

IPythonコンソール

```
In [5]: runfile('C:/Python/sample04.py', wdir='C:/Python')
[4, 7, 13]
[1, 4, 7]
[4, 7, 13, 17]
[4, 13]

In [6]:
```

スライス機能により作り出されたリスト

インデックスを2ずつ増やしているので、インデックス1の次が3になる

講師 スライスでは、インデックスを「いくつごとに」数えるかを決める「ステップ」を指定できます。また、開始インデックスや終了インデックスを省略することもできます。

愛 スライス…。

講師 リストのオブジェクトの要素数を調べたいときは、「len関数」を使います。このlen関数は、文字列の文字数を調べるときにも使えます。

進 文字列も、オブジェクトじゃからのう。

sample04.py

```
number = [1, 4, 7, 13, 17]

print(len(number))
print(len('ABCDEFG'))
```

numberの要素数を返す

文字列の文字数を返す事もできる

IPythonコンソール

```
In [7]: runfile('C:/Python/sample04.py', wdir='C:/Python')
5
7

In [8]:
```

numberの要素数は５個

'ABCDEFG'は７文字

講師 当然ですが、リストは文字列を格納したり、別のリストを格納することもできます。

sample04.py

```
strings = ['マダイ', 'ブリ', 'ヒラメ']

print(strings[0])
print(strings[1])
print(strings[2])

number = [[4,5],[9,7],[1,6]]

print(number[0])
print(number[1])
print(number[2])
```

要素が文字列のリスト

リストを保持するリスト（2次元リスト）

IPythonコンソール

```
In [9]: runfile('C:/Python/sample04.py', wdir='C:/Python')
マダイ
ブリ
ヒラメ
[4, 5]
[9, 7]
[1, 6]

In [10]:
```

文字列の要素が3個

各要素は、リスト

講師 もし、リストに新たな要素を追加したり、挿入したり、削除したりしたいときは、「appendメソッド」や「insertメソッド」、「removeメソッド」を使います。

進 「メソッド」ってなんじゃぁ?

講師 おっと、そうですね。説明を忘れていました。メソッドは、オブジェクトが持っている便利な関数です。「オブジェクト名.メソッド名()」のように記述して利用します。他の関数と同じように、括弧の中にデータを記述して、そのデータをメソッドに渡すことができます。例えば、appendメソッドは、リストの最後に要素を追加します。また、insertメソッドを使うと、

指定したインデックスの場所に要素を挿入できます。さらに、removeメソッドは、要素を左から検索して、指定された内容のオブジェクトを見つけたら削除します。

sample04.py

```
number = [10, 30]
print(number)

number.append(50)
print(number)

number.insert(1,20)
print(number)

number.remove(20)
print(number)
```

appendメソッドを使い、numberオブジェクトの最後に50を追加

インデックス1の場所に、20を挿入

要素を左から順番に調べて、20を見つけたら削除する

IPythonコンソール

```
In [11]: runfile('C:/Python/sample04.py', wdir='C:/Python')
[10, 30]
[10, 30, 50]
[10, 20, 30, 50]
[10, 30, 50]

In [12]:
```

50が追加された

インデックス1に、20が挿入された

20が削除された

講師 リストには、便利なメソッドがまだまだありますが、最後に「sortメソッド」を紹介しておきます。このメソッドは、要素を「昇順」で並べ替えます。括弧の中に「reverse = True」と記述すると、降順になります。

sample04.py

```
number = [3,6,4,2,8,9]
print(number)

number.sort()
print(number)

number.sort(reverse = True)
print(number)
```

要素を昇順に並べ替え

要素を降順に並べ替え

IPythonコンソール

```
In [13]: runfile('C:/Python/sample04.py', wdir='C:/Python')
[3, 6, 4, 2, 8, 9]
[2, 3, 4, 6, 8, 9]
[9, 8, 6, 4, 3, 2]

In [14]:
```

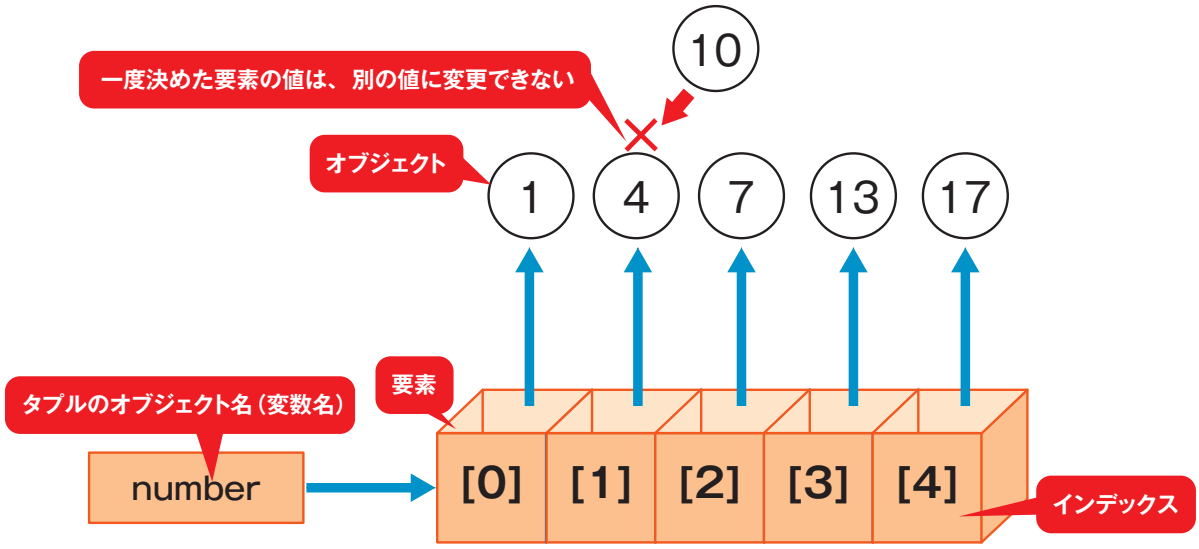
要素が昇順に並んだ

要素が降順に並んだ

愛 整列したオブジェクトは、美しい…。

講師 …。実は、リストと同じ構造を持った「タプル」というオブジェクトもあります。タプルを作るときは、括弧で要素を括ります。それ以外は、基本的にリストと同じ機能を持ちます。ただし、リストと異なり、要素を変更できないという大きな違いがあります（**図2**）。

図2●タプルのオブジェクトのイメージ



sample04.py

```
number = (1, 4, 7, 13, 17)
print(number)

number.insert(1,10)
print(number)
```

タプルの生成

タプルを変更しようとする

IPythonコンソール

```
In [15]: runfile('C:/Python/sample04.py', wdir='C:/Python')
(1, 4, 7, 13, 17)
Traceback (most recent call last):

  File "<ipython-input-6-171e6c8895d6>", line 1, in <module>
    runfile('C:/Python/sample04.py', wdir='C:/Python')

...
```

タプルの表示

エラーになる

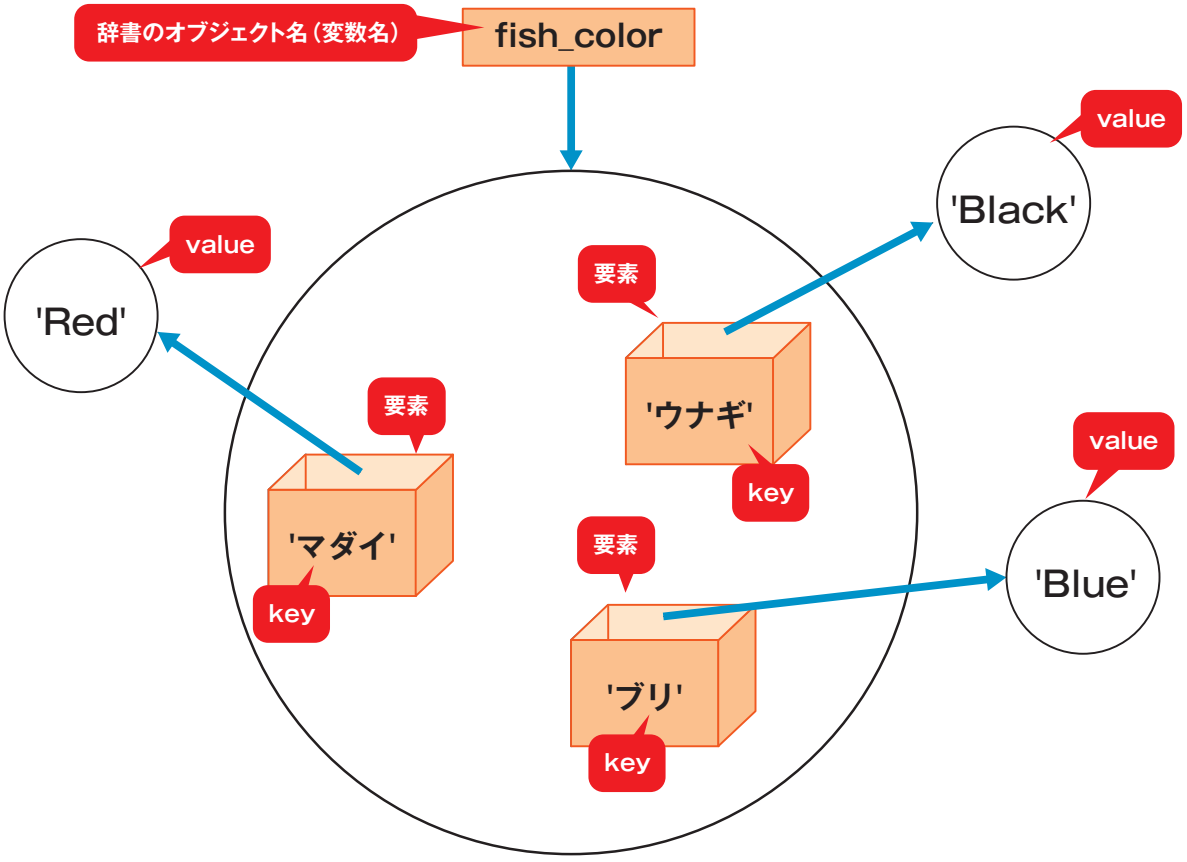
進 要素を変更できないって、意味あるのかのう?

講師 良い疑問ですね。そう思うのは自然です。プログラムでは、「変更したくないデータ」を扱うことが結構あります。そのような場合はリストではなくタプルを使います。また、タプルの方が変更できない分、リストよりも少しだけ高速に処理できます。

進 納得いきもうした!

講師 最後は「辞書」です。辞書は「連想配列」とも呼ばれるオブジェクトです。個々の要素は、順序付けのされていない「key」(キー)と「value」(バリュー)のペアで構成されています(**図3**)。リストやタプルでは、要素はインデックスという順序付けされた番号で管理していたので、ずいぶん違いますね。ちなみに、辞書では、keyが重複することは許されません。

図3●辞書のオブジェクトのイメージ



進 辞書が、名前を使ってデータを管理するオブジェクトってやつじゃろ。

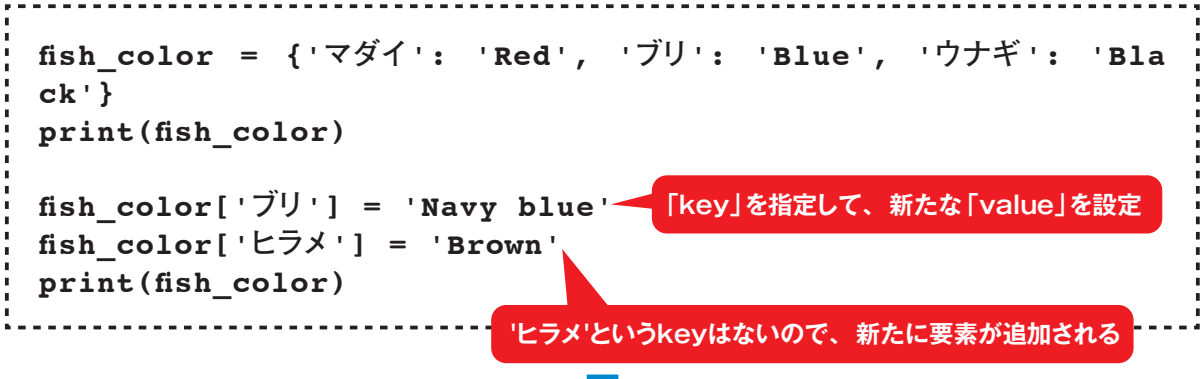
講師 そうです。keyは辞書が保持する要素の名前に相当します。辞書を作るときは、波括弧で括った中に、「key:value」のペアで要素を記述します。

sample04.py

IPythonコンソール

講師 要素は、リストのように角括弧の中にkeyを記述することで指定します。すでにあるkeyの要素を記述した場合、valueは上書きされます。新しいkeyの場合は新たに「key:value」のペアが追加されます。

sample04.py



IPythonコンソール

```
In [18]: runfile('C:/Python/sample04.py', wdir='C:/Python')
{'マダイ': 'Red', 'ブリ': 'Blue', 'ウナギ': 'Black'}
{'マダイ': 'Red', 'ブリ': 'Navy blue', 'ウナギ': 'Black', 'ヒラメ': 'Brown'}

In [19]:
```

Blue → Navy blueへ

ヒラメが追加されている

愛 仲間が、増えた…。うれしい…。

講師 今日は、ここまでにしましょう。明日は、いよいよオリジナルの関数を作ります。お楽しみに。

4日目

関数

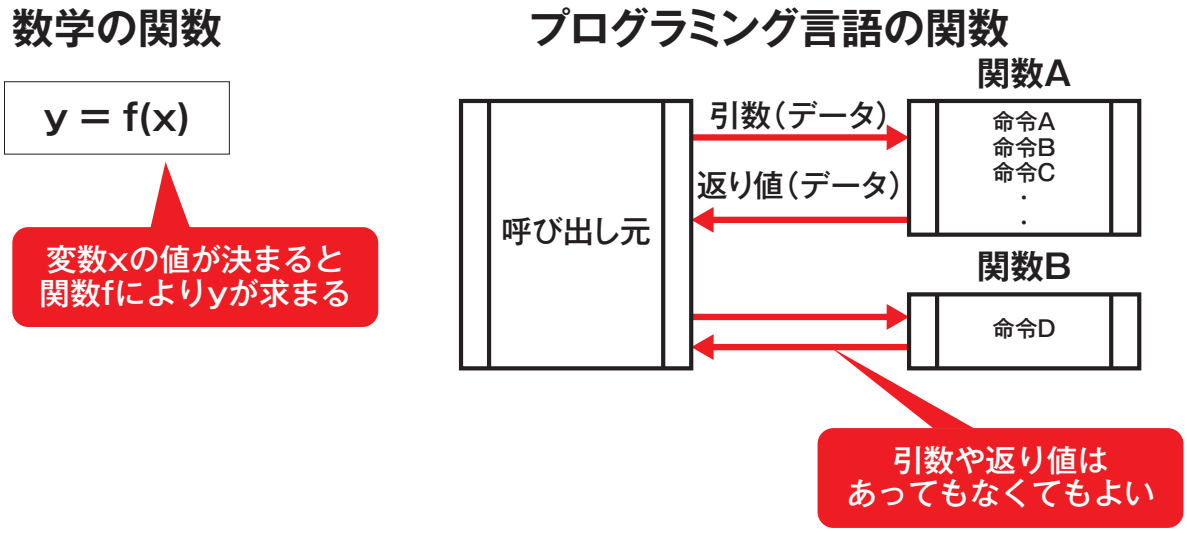
Part 1 関数の定義

講師 本日は、関数のお話です。

進 関数…数学の話ならまあまあ得意じゃ。

講師 紛らわしいのですが、プログラミングにおける「関数」は、数学の「関数」とは少し意味が異なります。プログラミングでは、単に命令や処理をまとめただけのものも「関数」と呼びます。ただし、「値」を「引数」（ひきすう）として与えることができ、かつ、値を「返り値」として返すこともできる、数学の関数と同じような関数もあります（**図1**）。ですから、数学の関数と同じ名前が付いているのです。

図1●関数のイメージ



愛 関数は、引数を食べて、返り値を排泄する…。

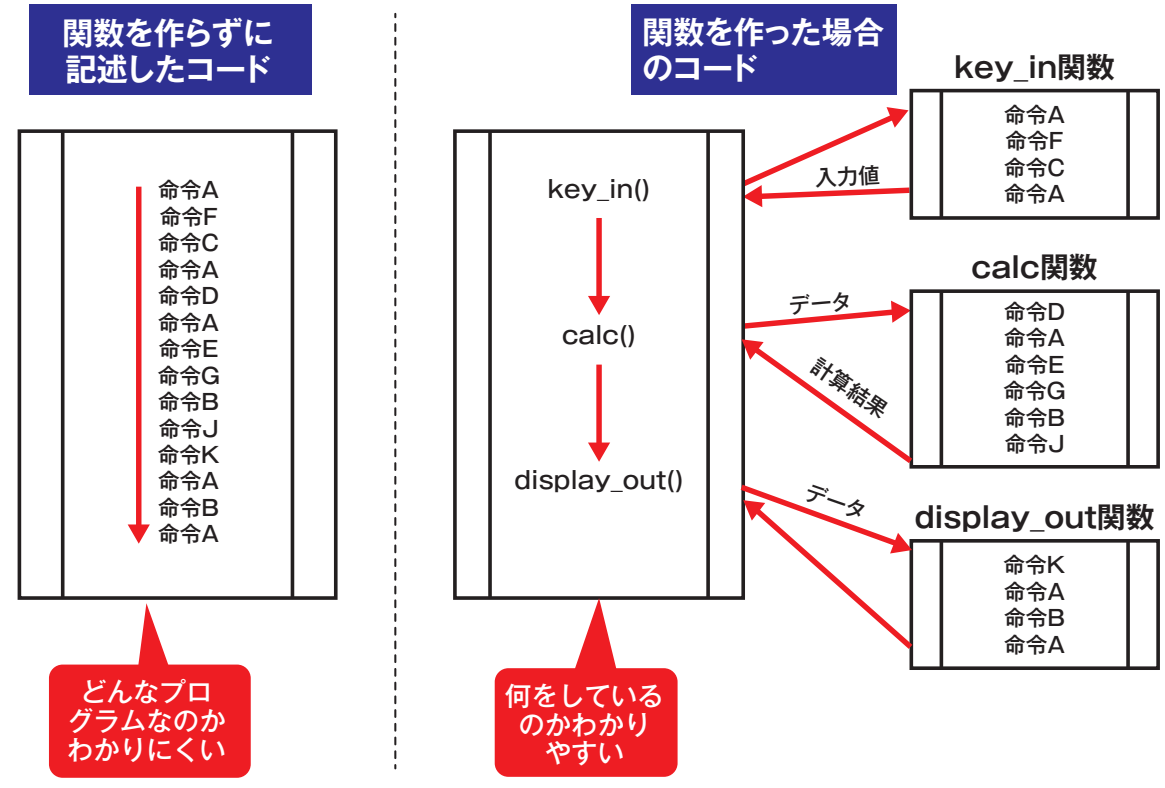
進 愛ちゃんは、いつも変わった例え方をするのう。

講師 プログラミングで関数を作る理由は、処理の流れをわかりやすくするためです。例えば、入力された数字を計算して画面に出力する処理を考えます。このような処理を、関数を作らずにプログラミングすることはできますが、ソースコードが読みにくくなり、処理内容も把握しづらくなります。一方、処理を機能ごとにkey_in関数、calc関数、display_out関数のように分けると、入力値を計算して出力するプログラムであることが一目でわかるようになります（**図2**）。

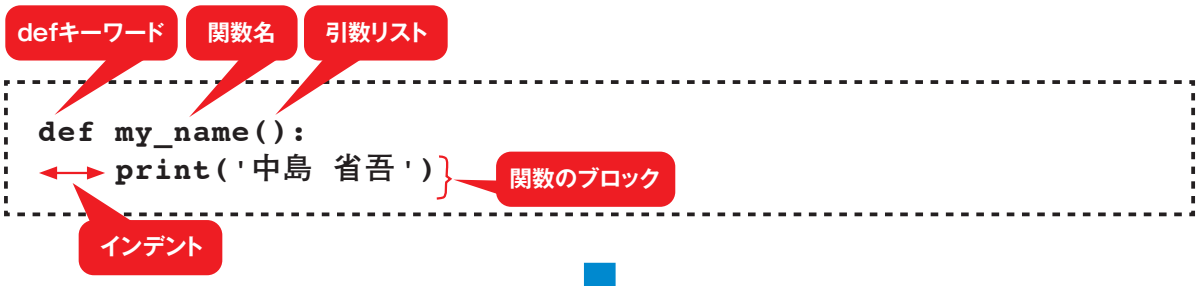
進 確かに、関数に分けた方がわかりやすいのう。

講師 それでは、Pythonで関数を作ってみましょう。関数は「def」というキーワードで定義します。defの後ろには、関数名、括弧の中の引数リスト、コロンと続きます。引数リストは、引数がなければ括弧だけを記述します。そして、if文やwhile文と同様、改行したらインデントを入れてブロックを作り、ブロック内に処理を記述します。例えば、自分の名前を表示するだけのmy_name関数を作って実行してみましょう。

図2●関数の利用



sample05.py



IPythonコンソール

```
In [1]: runfile('C:/Python/sample05.py', wdir='C:/Python')

In [2]:
```

実行しても、何も表示されない

進 なんじゃ、動かんぞ!

講師 関数は、定義しただけでは実行されません。実行するには、関数を呼び出す必要があります。関数を呼び出すには、関数名()のよう記述します。

sample05.py

```
def my_name():
    print('中島 省吾')

my_name()
```

my_name関数の呼び出し

関数の外から呼び出すのでインデントは付けない

IPythonコンソール

```
In [2]: runfile('C:/Python/sample05.py', wdir='C:/Python')
中島 省吾

In [3]:
```

my_name関数が実行された

愛 動いた…良い子…。

講師 …。…次に、関数にデータを渡してみましょう。この関数に渡すデータを「実引数」と呼びます。実引数を受け取るには、関数側に変数が必要です。この変数のことを「仮引数」と呼びます。仮引数の名前は、変数に使用できる名前であれば、どのようなものでも構いません。複数の仮引数がある場合は「,」（カンマ）で区切ります。次のコードは、数値を2つ渡すと、その合計を表示するadd関数です。引数を渡す順番に注意してください。

sample05.py

```
def add(x, y):
    print(x + y)

add(3, 4)
```

データを受け取るための変数（仮引数）

関数の呼び出し（実引数を関数に渡す）

IPythonコンソール

```
In [1]: runfile('C:/Python/sample05.py', wdir='C:/Python')
7
In [2]:
```

実行結果

講師 次に、値を呼び出し側へ返します。関数から値を返すには、return文を使います。return文の本来の役割は、処理を呼び出し元へ戻すことです。このreturn文の後ろに値を記述すると、その値が呼び出し元へ返ります。この値は、「返り値」や「戻り値」と呼ばれます。

sample05.py

```
def add(x, y):
    sum = x + y
    return sum

sum = add(3, 4)
print(sum)
```

合計値を呼び出し元へ返すreturn文

IPythonコンソール

```
In [4]:  runfile('C:/Python/sample05.py', wdir='C:/Python')
7

In [5]:
```

実行結果

講師 ここまでが、関数の基本です。さらにPythonの関数では、引数に「デフォルト値」を設定することができます。仮引数に=記号を付けてデフォルト値を設定します。

sample05.py

デフォルト値

```
def def_para(x = 10):
    print(x)

def_para()
```

何も渡さずに呼び出す

IPythonコンソール

```
In [6]:  runfile('C:/Python/sample05.py', wdir='C:/Python')
10

In [7]:
```

デフォルト値が使われる

講師 また、データを引数の名前（変数の名前）で指定して渡すこともできます。名前を指定して渡すことで、仮引数の順番に関係なく渡せます。

sample05.py

```
def def_para(x, y):
    print(x - y)

def_para(y=5, x=3)
```

データ（実引数）を変数名（引数名）で指定して渡す

IPythonコンソール

```
In [7]: runfile('C:/Python/sample05.py', wdir='C:/Python')
-2

In [8]:
```

xに3が、yに5が渡された結果−2になった

講師 最後に、ちょっと難しいのですが「可変長引数」についても説明しておきます。可変長引数とは、引数の数があらかじめ決まっていない引数のことで、1つの仮引数で複数のデータを受け取ることができます。

進 1つの変数で、複数のデータを受け取る…どうやるんじゃ?

講師 可変長引数の仮引数は、データをタプルや辞書の要素として受け取ります。例えば、仮引数に「*」（アスタリスク）を付けると、渡されたデータをタプルの要素として受け取ります。また、「**」を付けた場合は、辞書の要素として受け取ります。

データをタプルの要素として受け取る

sample05.py

```
def def_para_t(*x):
    print(x)

def def_para_d(**x):
    print(x)

def_para_t(100, 'Hello', 3.14)

def_para_d(x = 100, str = 'Hello', y = 3.14)
```

データを辞書の要素として受け取る

3つのデータを渡す

3つのデータを、引数名を指定して渡す

IPythonコンソール

```
In [9]: runfile('C:/Python/sample05.py', wdir='C:/Python')
(100, 'Hello', 3.14)
{'x': 100, 'str': 'Hello', 'y': 3.14}

In [10]:
```

実引数がタプルにまとめられている

実引数が辞書にまとめられている

進 なるほど。実引数をタプルや辞書に変換して受けとるんじゃの。

講師 それでは、休憩をいれて「モジュール」へ進みましょう。

4日目 Part 2 モジュール

講師 便利な関数をたくさん作ったら、誰かに使ってもらいたいですよね。そんなとき、複数の関数を1つの「モジュール」にまとめることができます。「モジュール」とは、Pythonのコードが記述されているファイルのことで、import（インポート）することで、便利な関数やオブジェクトを利用できるようになります。また、有志が作った様々な機能がモジュールとしてインターネットで公開されています。そうしたモジュールの多くは、無償でダウンロードできます。AnacondaにもAIプログラム開発用をはじめとした様々なモジュールが、あらかじめインストールされているんですよ。

愛 私もimportして、機能を追加する…。

進 なぜか、愛ちゃんならできそうな気がするのう。

講師 まず、簡単な関数をモジュールにしてみましょう。Spyderの「ファイル」メニューから「新規作成」を選択して、エディターで「名前を表示するmy_name関数」、「2つの引数の合計を返すadd関数」を定義してください。また、関数が正しく定義されていることを確認するため、その関数を呼び出すコードも追加しておきます。できたら、実行してみましょう。

mymodule.py

```
def my_name():
    print("中島省吾")

def add(x, y):
    return x + y

my_name()
print(add(3, 4))
```

名前を表示するmy_name関数

2つの引数の合計を返すadd関数

関数の呼び出し

IPythonコンソール

```
In [1]:  runfile('C:/Python/mymodule.py', wdir='C:/Python')
中島省吾
7

In [2]:
```

my_name関数が実行された

add関数が実行された

講師 実行できたら、mymodule.pyで保存します（**図1**）。

講師 このモジュールを、sample06.pyでimportして利用してみましょう。

sample06.py

```
import mymodule
```

mymodule.pyのインポート

IPythonコンソール

```
In [1]:  runfile('C:/Python/sample06.py', wdir='C:/Python')
中島省吾
7

In [2]:
```

実行すると、my_name関数とadd関数が実行された

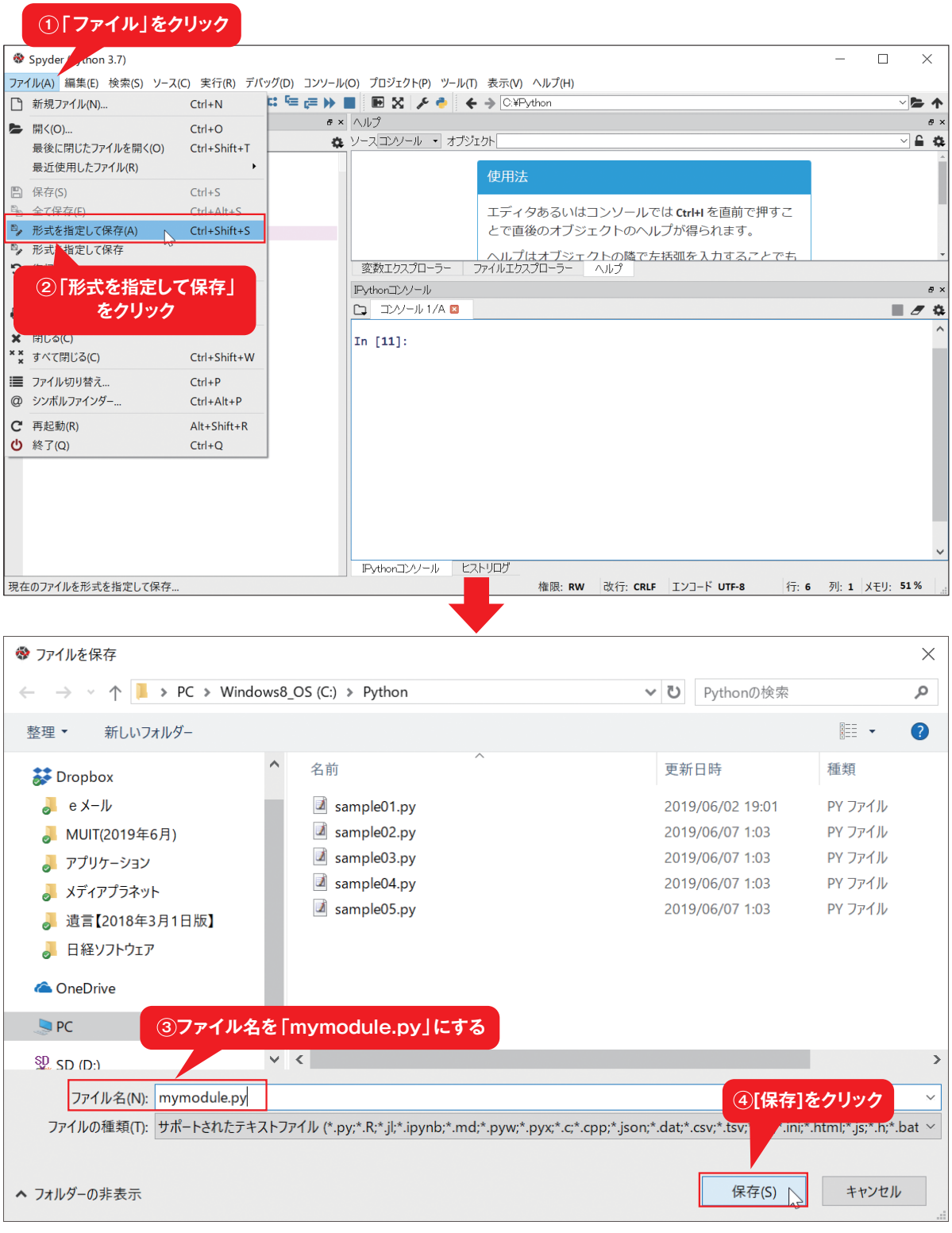
講師 import文が実行されると、モジュール内の関数が呼び出され、実行結果が表示されます。今度は、importした関数を呼び出してみます。

sample06.py

```
import mymodule

mymodule.my_name()
print(mymodule.add(3, 4))
```

importされた関数を呼び出す

図1●mymodule.pyで保存



IPythonコンソール

```
In [1]:  runfile('C:/Python/sample06.py', wdir='C:/Python')
Reloaded modules: mymodule
中島省吾
7
中島省吾
7

In [2]:
```

importによる関数の実行

mymoduleモジュールの関数を呼び出した結果

講師 実行すると、importした関数を呼び出せていることがわかります。

進 モジュール内の関数定義だけimportして、関数呼び出しを無視することはできないの?

講師 それには、モジュール内に「if __name__ == "__main__":」のブロックを追加します。このif文を利用すると、関数の定義だけを行い、関数呼び出しをスルーできます。mymodule.pyを次のように書きかえます。

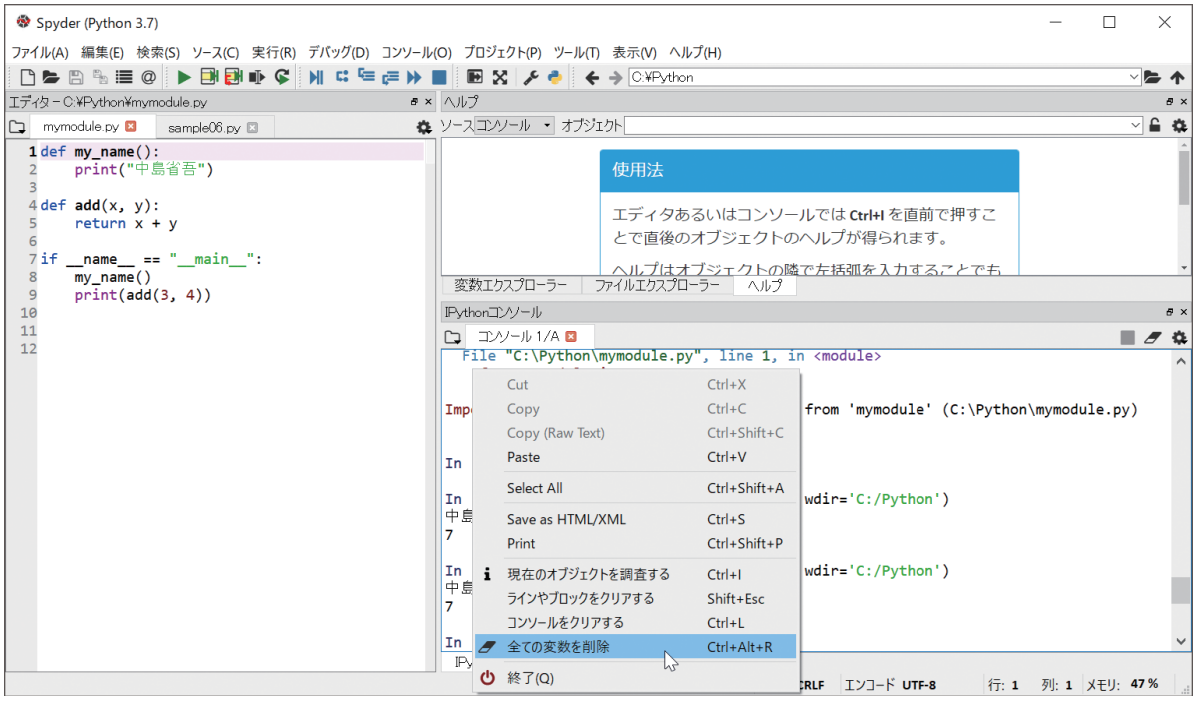
mymodule.py

```
def my_name():
    print("中島省吾")

def add(x, y):
    return x + y

if __name__ == "__main__":
    my_name()
    print(add(3, 4))
```

importされるときはスルーされるように追加

講師 次に、IPythonコンソール上で右クリックし、「全ての変数を削除」を実行してから、sample06.pyのコードを再び実行します（**図2**）。

図2●「全ての変数を削除」する



sample06.py

```
import mymodule

mymodule.my_name()
print(mymodule.add(3, 4))
```

IPythonコンソール

```
In [1]: runfile('C:/Python/sample06.py', wdir='C:/Python')
Reloaded modules: mymodule
中島省吾
7

In [2]:
```

モジュール内の関数呼び出しは実行されない

進 こうすると、実行したときとimportしたときとで、挙動を分けることができるんじゃな。

講師 ハイ。importの方法にはもう1つ、fromでimportする方法もあります。fromを使うと、モジュール名を使わずに、直接関数を呼び出すことができます。

sample06.py

```
from mymodule import my_name, add

my_name()
print(add(3, 4))
```

mymoduleモジュールのmy_name関数とadd関数をインポート

モジュール名なしで呼び出せる

IPythonコンソール

```
In [1]: runfile('C:/Python/sample06.py', wdir='C:/Python')
中島省吾
7

In [2]:
```

my_name関数が実行された

add関数が実行された

Part 3 4日目 組み込み関数を使う

講師 Pythonには、すでに用意されているオブジェクトや関数がたくさんあります。このような関数を「組み込み関数」と呼びます。次に、組み込み関数の一覧を示します（表1）。

表1●組み込み関数の一覧

関数名	機能
abs()	引数の絶対値を返す
all()	イテラブルオブジェクトの全要素が真（もしくは空）なら True を返す
any()	イテラブルオブジェクトのいずれかの要素が真なら True を返す。空なら False を返す
ascii()	オブジェクトの印字可能な文字列を返す
bin()	整数を2進文字列に変換する
bool()	引数のブール値を返す
breakpoint()	デバッガのブレークポイント
bytearray()	引数のバイト配列を返す
bytes()	引数のbytes オブジェクトを返す
callable()	引数が呼び出し可能オブジェクトであれば True、そうでなければ False を返す
chr()	Unicode 文字を表す文字列を返す
classmethod()	メソッドをクラスメソッドへ変換する
compile()	引数を AST オブジェクトにコンパイルする
complex()	文字列や数を複素数に変換する
delattr()	指名された属性を削除する
dict()	新しい辞書を作成する

（表1の続き）

関数名	機能
dir()	オブジェクトの属性リストを返す
divmod()	整数の除算を行ったときの商と剰余を返す
enumerate()	イーナムレイトオブジェクトを返す
eval()	文を式として評価する
exec()	文を実行する
filter()	引数の条件に沿う要素のみを抽出する
float()	数または文字列から浮動小数点数を生成する
format()	引数を書式化された表現に変換する
frozenset()	新しいfrozensetオブジェクトを返す
getattr()	オブジェクトの属性値を返す
globals()	現在のグローバルシンボルテーブルを辞書で返す
hasattr()	引数がオブジェクトの属性名ならTrue、違う場合はFalseを返す
hash()	オブジェクトのハッシュ値を返す
help()	ヘルプシステムを起動する
hex()	引数を16進数で表現する
id()	オブジェクトのIDを返す
input()	入力から1行読み込み、文字列に変換して返す
int()	数値または文字列を整数オブジェクトに変換する

（表１の続き）

関数名	機能
isinstance()	引数がインスタンス、もしくはサブクラスのインスタンスかを調べる
issubclass()	引数がサブクラスかどうかを調べる
iter()	引数のイテレータオブジェクトを返す
len()	オブジェクトの要素数を返す
list()	リストを生成する
locals()	現在のローカルシンボルテーブルを表す辞書を更新して返す
map()	すべての要素に適用するイテレータを返す
max()	最大の要素、または２つ以上の引数の中で最大のものを返す
memoryview()	オブジェクトの「メモリービュー」オブジェクトを返す
min()	最小の要素、または２つ以上の引数の中で最小のものを返す
next()	次の要素を取得する
object()	新しいオブジェクトを返す
oct()	整数を８進文字列に変換する
open()	ファイルを開き、そのファイルオブジェクトを返す
ord()	１文字のUnicode文字のUnicodeコードポイントを表す整数を返す
pow()	累乗を返す
print()	引数を標準出力に出力する
property()	プロパティ属性を返す

（表1の続き）

関数名	機能
range()	連続した数字オブジェクトを生成する
repr()	オブジェクトの文字列を返す
reversed()	要素を逆順に取り出すイテレータオブジェクトを返す
round()	小数部を丸めた値を返す
set()	新しいセットオブジェクトを返す
setattr()	値を属性に関連付ける
slice()	スライスされたオブジェクトを返す
sorted()	要素を並べ替えた新たなリストを返す
staticmethod()	メソッドを静的メソッドへ変換する
str()	数値を文字列オブジェクトに変換する
sum()	要素を左から右へ合計し、総和を求める
super()	親クラスにメソッドの呼び出しを委譲する
tuple()	タプルを生成する
type()	オブジェクトの型を返す
vars()	モジュール、クラス、インスタンスの __dict__ 属性を返す
zip()	オブジェクトの要素を集めたイテレータを生成する
__import__()	モジュールをインポートする

進 こりゃいったい、何個あるんじゃ。

講師 Anacondaでは、さらに多くのオブジェクトや関数がモジュールとして提供されています。ただ、ここですべての関数を紹介することはできないので、書式化のためのformat関数と、連続した数値を生成するrange関数の2つを紹介しましょう。

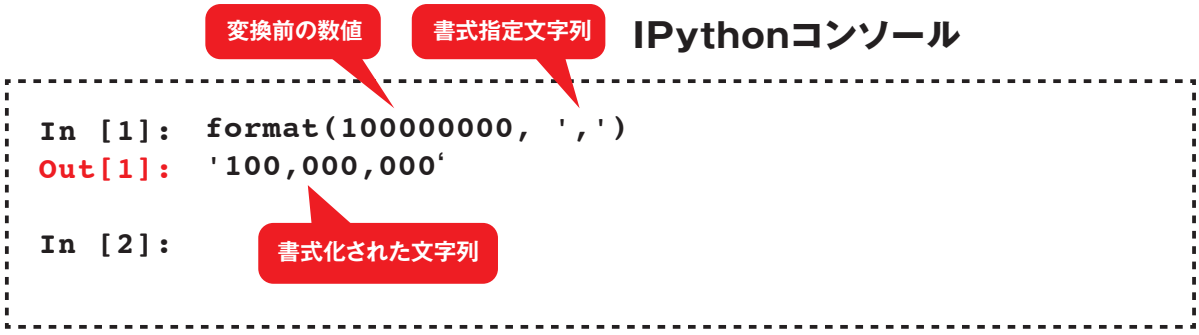
愛 …ザンネン…。

講師 最初に紹介するformat関数は、書式化のための関数です（表2）。

表2●format関数

format(value, format_spec)	
引数	説明
value	変換前の値（文字列や数値など）
format_spec	書式指定文字列
返り値	書式化された文字列

講師 引数には、変更前の文字列と書式を指定します。この書式を決める文字列を「書式指定文字列」といいます。例えば、数値を3桁のカンマ区切りの文字列に変換したければ、書式指定文字列にカンマを指定します。IPythonコンソールで確認してみましょう。



講師 実際の書式指定文字列には、次のような種類があり、組み合わせて利用します（表3）。

表3●書式指定文字列

書式指定文字列	意味
'<'	左詰め（ほとんどのデフォルト）
'>'	右詰め
'='	符号の後ろを埋める（数値型に対してのみ有効）
'^'	中央寄せ

（表3の続き）

書式指定文字列	意味
'+'	符号を、正数、負数の両方に指定する
'-'	符号を、負数に対してのみ指定する（デフォルト）
空白	正数なら空白を前に付け、負数なら負号を前に付ける
','	千の位の区切り文字
'_'	浮動小数点数、整数の千倍ごとの区切り文字
's'	文字列
'b'	2進数
'c'	文字
'd'	10進数
'o'	8進数
'x'	16進数（小文字）
'X'	16進数（大文字）
'e'	指数を示す 'e' を使って指数表記
'E'	指数を示す 'E' を使って指数表記
'f'	固定小数点数表記（小文字）。デフォルトの精度は 6
'F'	固定小数点数表記（大文字）。デフォルトの精度は 6
'%'	固定小数点数フォーマットでパーセント付き

講師 また、format関数と同じ機能の「formatメソッド」もあります。Pythonでは、関数は単体のオブジェクトですが、メソッドはオブジェクトが持つ関数のことです。

進 関数はオブジェクトで、オブジェクトの関数がメソッド…。相変わらずややこしいのう。

講師 変換前の文字列の最後に「.」（ドット）を入力してformatメソッドを記述します。変換する値はformatメソッドの引数に渡します。変換前の文字列の中に{ }（波括弧）とその中に「:」（コロン）を書いて書式指定文字列を記述します。こうすると、{ }の部分を書式変換した文字列で置き換えることができます。この波括弧と中の文字列を「フォーマットフィールド」と呼びます。

愛 フォーマットフィールド、全開!!!

進 うわぁ、ビックリした。愛ちゃん、どうしたんじゃ。

講師 さ、栄井さん…大丈夫…そうですね。え～と、formatメソッドの使い方でしたね。例えば、数値をカンマで3桁区切りの文字列に変換するなら、formatメソッドでは次のようにします。

書式指定文字列　変換前の数値

IPythonコンソール

```
In [2]:  '{:,}'.format(100000000)
Out[2]:  '100,000,000'

In [3]:
```

書式化された文字列

講師 右寄せや左寄せなどの、文字の位置決めもできます。

IPythonコンソール

```
In [3]:  '左詰め : {:<10}'.format(3)
Out[3]:  Out[4]: '左詰め : 3          '

In [4]:  '右詰め : {:>10}'.format(3)
Out[4]:  Out[5]: '右詰め :          3'

In [5]:  '右詰め : {:>10}'.format(-3)
Out[5]:  Out[6]: '右詰め :         -3'

In [6]:  '右詰め : {:0=+10}'.format(3)
Out[6]:  Out[7]: '右詰め : +000000003'

In [7]:  '右詰め : {:@>10}'.format(3)
Out[7]:  Out[8]: '右詰め : @@@@@@@@@3'

In [8]:  '中央 : {:^10}'.format(3)
Out[8]:  Out[9]: '中央 :     3      '

In [9]:  '中央 : {:^10}'.format(3.14)
Out[9]:  Out[10]: '中央 :    3.14    '
```

空白10文字分の左詰め

空白10文字分の右詰め

空白10文字分の右詰め

0で埋める
（+記号は10文字に含まれる）

文字で埋める

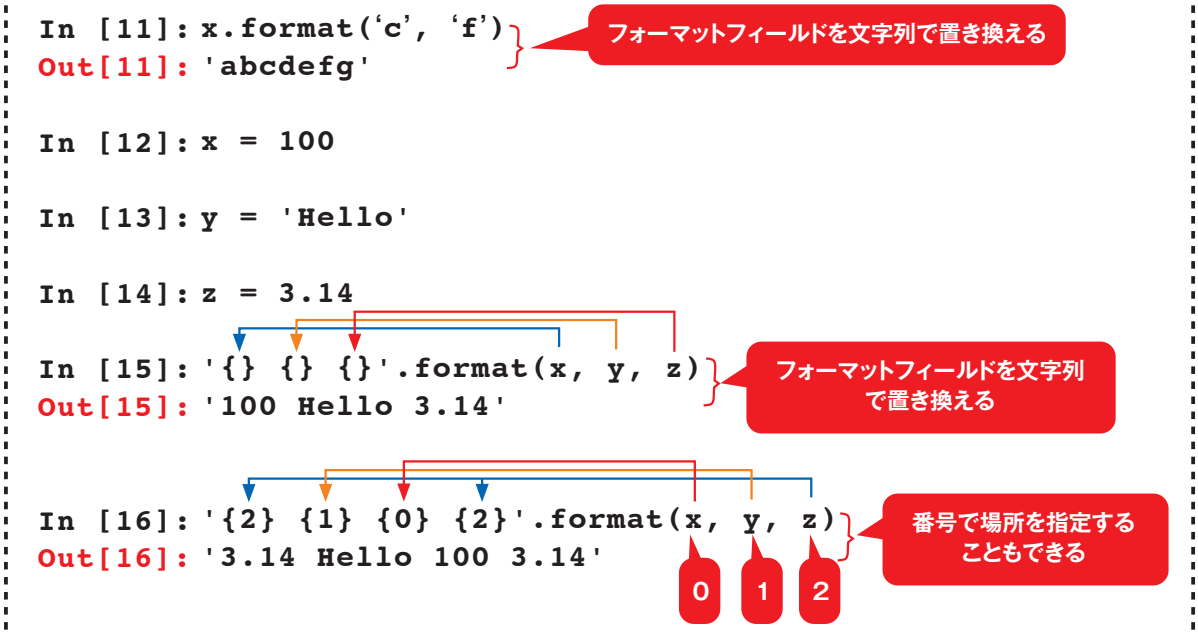
空白10文字分の中央ぞろえ

空白10文字分の中央ぞろえ
（小数点も含まれる）

講師 フォーマットフィールドは、文中に複数指定できます。また、数字を使用して置き換える場所を決めることもできます。

IPythonコンソール

```
In [10]: x = 'ab{}de{}g'

In [11]: x.format('c', 'f')
Out[11]: 'abcdefg'

In [12]: x = 100

In [13]: y = 'Hello'

In [14]: z = 3.14

In [15]: '{} {} {}'.format(x, y, z)
Out[15]: '100 Hello 3.14'

In [16]: '{2} {1} {0} {2}'.format(x, y, z)
Out[16]: '3.14 Hello 100 3.14'
```



講師 最後に、range（レンジ）関数です。range関数は、数列を作り出す関数です（**表4**）。

表4●range関数

range(start, stop[, step])	
引数	説明
start	数列を開始する値（指定しない場合は0）
stop	終了する値（この値は含まれない）
step	ステップ数（省略した場合は1）
返り値	数列

講師 次のコードをエディターに入力して実行してみましょう。このコードでは、開始の値を1、終了の値を5にしているので、1から4までの連続した数値を生成します。

1から4までの連続した数値を生成する

sample07.py

```
for i in range(1, 5):
    print(i)
```

インデント

IPythonコンソール

```
In [17]: runfile('C:/Python/sample07.py', wdir='C:/Python')
1
2
3
4

In [18]:
```

生成された数列

講師 range関数のstep引数を利用すれば、3ずつ増える数列や、5ずつ増える数列などを作ることもできます。

sample07.py

```
for i in range(10, 20, 5):
    print(i)
```

10から始まり19まで5ずつ増える数列を生成

IPythonコンソール

```
In [18]: runfile('C:/Python/sample07.py', wdir='C:/Python')
10
15

In [19]:
```

生成された数列

講師 今日は、ここまでにしましょう。お疲れさまでした。

5日目

クラスとオブジェクト

5日目 Part 1 オブジェクトとは

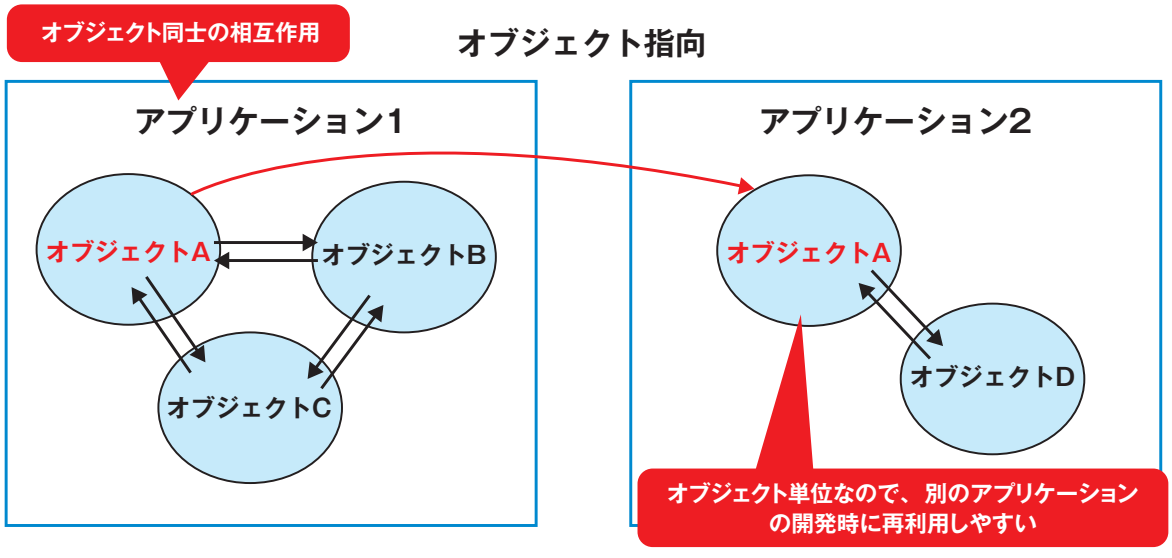
講師 Pythonはオブジェクト指向言語ですが、そもそも「オブジェクト指向プログラミング」って、どんなプログラミングだと思いますか。

進 「もの指向・・・」って言われても、ようわからん。

愛 オブジェクト指向プログラミングは、アプリケーションを「“データ”と“操作”で構成されるオブジェクト」同士の相互作用でプログラミングする手法…。

講師 おおぉ、さすが栄井さん。その通りです。オブジェクト指向では、プログラムをオブジェクト単位で考えます。そのオブジェクト同士が、お互いを利用しあうことでアプリケーションが動作するようにプログラムを開発する手法が「オブジェクト指向」ですね。そうすることで、プログラムの再利用性やメンテナンス性が向上すると言われています（**図2**）。

図1●オブジェクト指向



進 愛ちゃんは「オブジェクト」がどんなものかわかっとるようじゃだけど、わしにはサッパリじゃ。

愛 ごめんなさい…。こういうときどんな顔すればいいかわからない…。

講師 …。オブジェクトとは、現実世界の「もの」のことですが、本当の「もの」は複雑過ぎてプログラムでは表現できません。そこで、オブジェクトを「データ」と「操作」の2つに単純化します。Pythonでは「データは変数」、「操作は関数（メソッド）」で表現しますが、どちらも「オブジェクトのattribute（アトリビュート。属性）」と考えます。つまり、Pythonにとっては、データも操作も、「オブジェクトが持つ属性」というわけですね（図2）。

進 つまり、オブジェクトは変数や関数をまとめたものってことじゃな。

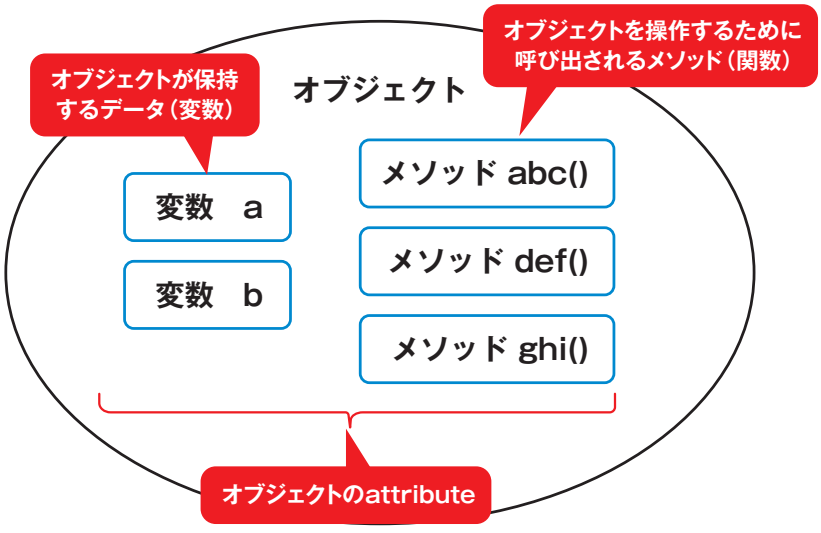
講師 だいたい、そのイメージで合っています。そして、オブジェクト指向では、オブジェクトは「クラス」から生成します。クラスは言わば「オブジェクトの設計図」ですね。クラスからオブジェクトを生成することで、同じオブジェクトを量産することが可能になります（図3）。

愛 私と…同じ…。

進 愛ちゃん、どないしたんじゃ。今日は少しおかしいで。

講師 また、オブジェクト指向には「継承」という概念もあります。継承とは、既にあるクラスを利用して新たなクラスを作る手法のことです（図4）。このような手法を「差分プログラミング」と呼ぶこともあります。

図2●オブジェクトのイメージ



講師 継承では、既存のクラスを「スーパークラ

図３●クラスからオブジェクトを生成する

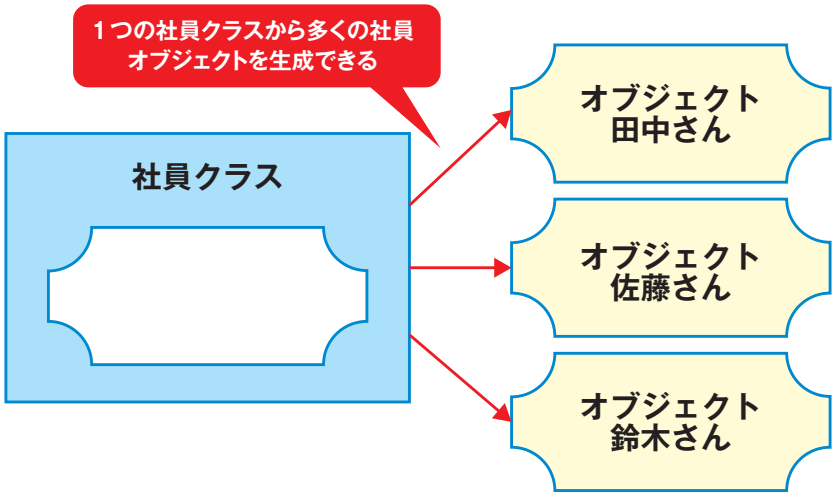
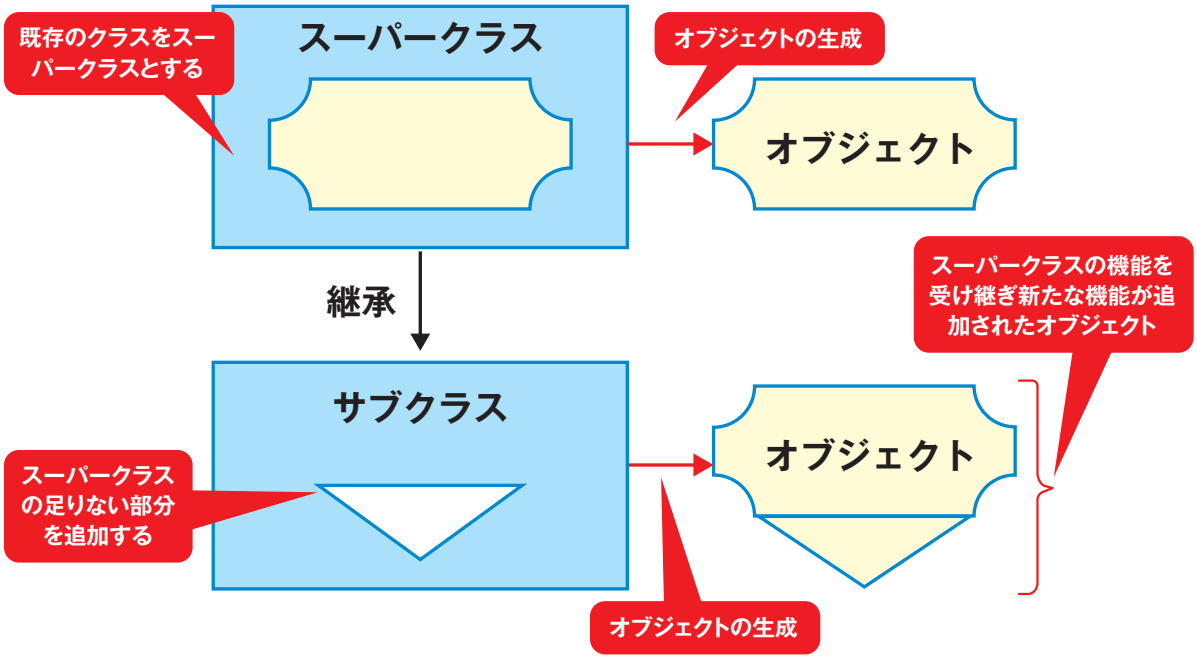


図４●継承によるクラスの再利用



ス」、機能を追加して新しく定義するクラスを「サブクラス」と呼びます。大規模なクラスをたくさん作るときは、抽象的なスーパークラスから複数の具体的なサブクラスを作るようにします。その方が、個々のクラスの見通しが良くなり、メンテナンスしやすいクラスにすることができます。それでは、休憩しましょう。次は、クラス定義へ進みます。

5日目 Part 2 クラス

講師 オブジェクト指向では、オブジェクトは「クラス」から生成します。そのため、オリジナルのオブジェクトを作るには、クラスを定義する必要があります。

進 クラスは、オブジェクトの設計図じゃからな。

講師 Pythonでクラスを定義するにはclassキーワードを使います。クラス名の後ろに「:」を付けて改行し、ブロックを作ります。その中で、オブジェクトの変数やメソッドを定義していきます。

講師 それでは、Employee（エンプロイー）という「社員」を表すクラスを作りましょう。社員の名前は「小鯖」さんにして、workメソッドを持つことにします。

sample07.py

```
class Employee:
    name = '小鯖　進'
    def work(self):
        print('一生懸命、働きます。')

m = Employee()
print(m.name)
m.work()
```

Employeeクラスの定義

Employeeオブジェクトにmという名前を付ける

オブジェクトのattributeを利用する

IPythonコンソール

```
In [1]: runfile('C:/Python/sample07.py', wdir='C:/Python')
小鯖　進
一生懸命、働きます。

In [2]:
```

オブジェクトの変数を参照した結果

オブジェクトのメソッドを呼びだした結果

進 わしのオブジェクトじゃ。

講師 クラスを定義するとき注意する点は、メソッドに第1引数を用意する場合と用意しない場合があることです。通常、第1引数を用意する場合は「self」という名前にします。そして、クラスを定義し終わったら、クラス定義のブロックの外で、クラス名()のように記述するとオブジェクトを生成できます。このオブジェクトにはまだ名前がないので、変数に割り当てることでオブジェクトに名前を付けます。

愛 オブジェクトが…生まれた…。

講師 実は、クラスを定義しただけでも「クラスオブジェクト」は生成されています。クラスオブジェクトには「クラス名」という名前があるので、そのまま「クラス名.変数名」や「クラス名.メソッド名()」で利用することができます。この場合のメソッドには、self引数は設定しないので注意しましょう。

sample07.py

```
class Employee:
    name = '小鯖　進'
    def work():
        print('一生懸命、働きます。')

print(Employee.name)
Employee.work()
```

クラスオブジェクトのメソッドには、引数を付けない

「クラス名.クラス変数」、
「クラス名.クラスメソッド()」でアクセスできる

IPythonコンソール

```
In [2]: runfile('C:/Python/sample07.py', wdir='C:/Python')
小鯖　進
一生懸命、働きます。

In [3]:
```

進 なんじゃなんじゃ? Pythonのオブジェクトは、名前を付けるオブジェクトと、クラスオブジェクトの2種類あるんかい。

講師 そうです。クラスオブジェクトは、クラス名ですぐに利用できますが、1つしか作ることができません。対して名前を付けるオブジェクトは、__init__メソッドを利用して同じようなオブジェクトを量産できます。次のコードで確認しましょう。

sample07.py

```
class Employee:
    def __init__(self, n):
        self.name = n

    def work(self):
        return '一生懸命、働きます。'

m = Employee('小鯖　進')
f = Employee('栄井　愛')

print(m.name)
print(m.work())

print(f.name)
print(f.work())
```

__init__メソッド（コンストラクタ）

②オブジェクトが保持する変数を初期化

①__init__メソッドに初期値を渡す

IPythonコンソール

```
In [3]: runfile('C:/Python/sample07.py', wdir='C:/Python')
小鯖　進
一生懸命、働きます。
栄井　愛
一生懸命、働きます。

In [4]:
```

コンストラクタにより初期化された名前

コンストラクタにより初期化された名前

進 1つのクラスから、2つのオブジェクトが生まれたのじゃな。

講師 ここでオブジェクトを初期化している「__init__メソッド」は、「コンストラクタ」と呼ばれています。コンストラクタを利用するには、クラスからオブジェクトを生成する際、括弧の中にデータを渡します。データは__init__メソッドに渡り、オブジェクトが保持する変数で管理されます。

進 なるほど、selfという変数は、自分のオブジェクトに付ける名前ってことじゃな。

講師 その通り。こうして、Pythonでは、オブジェクト指向を実現しています。最後に、継承の仕方だけ簡単に解説しておきます。継承とは、前に言ったように、既存のクラスを再利用する手法です。コードで確認しましょう。

sample08.py

```
class Parent:
    def p_method(self):
        print('スーパークラスのメソッド')

class Child(Parent):
    def s_method(self):
        print('サブクラスのメソッド')

c = Child()
c.p_method()
c.s_method()
```

スーパークラス

スーパークラス名

サブクラス

サブクラスのオブジェクトからスーパークラスのオブジェクトを利用できる

IPythonコンソール

```
In [4]: runfile('C:/Python/sample08.py', wdir='C:/Python')
スーパークラスのメソッド
サブクラスのメソッド
```

サブクラスのオブジェクトからスーパークラスのオブジェクトを利用できる

講師 既存のクラスがParentクラス、Parentクラスに機能を追加したクラスがChildクラスです。Childクラスの定義では、スーパークラスにParentクラスを指定しています。このように継承を利用すると、既存のクラスに変更を加えることなく、機能を拡張した新しいクラスを作ることができます。

5日目 Part 3 Pythonのエラー

講師 Pythonでは、エラーもオブジェクトです。エラーには2種類あります。1つは「Syntax Error」（シンタックスエラー）、つまり「構文エラー」ですね。もう1つは「Exception」（エクセプション）、これは「例外」ですね。

進 なんじゃ、ジンベイザメ級のヤツが出てきたぞい。

講師 言葉にすると難しそうですが、どちらも「エラー」という意味では同じです。まず、最初にSyntax Errorから説明していきます。Syntax Errorは、Pythonの構文ルールに従っていないコードに対して出力されるエラーです。例えばif文やwhile文では、ブロックを作るために「:」を記述しますが、このコロンを忘れると「SyntaxError: invalid syntax」になります。

IPythonコンソール

```
In [5]: if True
 File "<ipython-input-1-850f1eccbd09>", line 1
    if True
           ^
SyntaxError: invalid syntax
```

末尾のコロンを付け忘れた

SyntaxErrorになる

講師 また、Pythonはブロックをインデントで表現するので、インデントを忘れてもエラーになります。インデント忘れは「IndentationError」です。

IPythonコンソール

```
In [6]: if True:
        print('エラー')

 File "<ipython-input-2-5ddab6f3a54b>", line 2
    print('エラー')
        ^
IndentationError: expected an indented block
```

行頭のインデントを付け忘れた

IndentationErrorになる

愛 構文は…私が…私が守る。

進 愛ちゃん、そこまで思いつめんでもええじゃろ。

講師 もう1つのエラーは、「例外」です。例外には、例えば、数値を0で割ると発生する

「ZeroDivisionError」（ゼロディビジョンエラー）があります。また、変数や関数などのオブジェクトが見つからないときに発生する例外の「NameError」（ネームエラー）や、データ型が異なるときに発生する例外の「TypeError」（タイプエラー）などがあります。

IPythonコンソール

```
In [7]: 3 / 0
Traceback (most recent call last):

  File "<ipython-input-14-e1965806ec03>", line 1, in <modu
le>
    3 / 0

ZeroDivisionError: division by zero
```

0による割り算

ZeroDivisionErrorが発生

IPythonコンソール

```
In [8]: print(test)
Traceback (most recent call last):

  File "<ipython-input-15-4ddfce83ccd5>", line 1, in <modu
le>
    print(test)

NameError: name 'test' is not defined
```

初期化されていない未定義の変数

NameErrorが発生

IPythonコンソール

```
In [9]: 3 + '4'
Traceback (most recent call last):

  File "<ipython-input-17-580430704c79>", line 1, in <modu
le>
    3 + '4'

TypeError: unsupported operand type(s) for +: 'int' and
'str'
```

数値型と文字列型の加算

TypeErrorが発生

講師 このような例外は、様々なオブジェクトとしてあらかじめ定義されているため、「組み込み例外」と呼ばれます。例外をそのまま放置すると、プログラムは強制終了してしまいます。しかし、「例外処理」を行うことで強制終了を回避できます。例外処理には「try節」を使います。try節に、例外が発生する可能性のあるコードを書きます。そして、本当に例外が発生したら、発生したコードの下へは移行せず、例外が一致する「except節」へジャンプします。次のコードでは、3を0で割っているので、ZeroDivisionErrorが発生し、ZeroDivisionErrorのexcept節へジャンプしています。except節が実行されると、プログラムは強制終了しません。

try節の中に、例外が発生する可能性のあるコードを書く

sample07.py

```
try:
    print('3を0で割ります。')
    x = 3 / 0
    print('エラーが発生します。')
except ZeroDivisionError:
    print("ZeroDivisionErrorに対応しました。")
```

0で割り算を行いZeroDivisionError を発生させる

上のコードでエラーになると、このコードは実行されない

IPythonコンソール

```
In [10]: runfile('C:/Python/sample07.py', wdir='C:/Python')
3を0で割ります。
ZeroDivisionErrorに対応しました。
```

発生した例外がexcept節の例外と同じならコードが実行される

講師 except節は、発生するエラーごとに用意します。例えば、ZeroDivisionErrorとNameErrorが発生する可能性のあるコードでは、次のように2つのexcept節で対応します。

sample07.py

```
try:
    x = input('0か、1を入力:')
    print(3 / int(x))
    print(test)
    print('正常終了')
except ZeroDivisionError:
    print("ZeroDivisionErrorに対応しました。")
except NameError:
    print("NameErrorに対応しました。")
```

0を入力するとZeroDivisionErrorになる

初期化されていない変数を使用してNameErrorにする

IPythonコンソール

```
In [11]: runfile('C:/Python/sample07.py', wdir='C:/Python')

0か、1を入力:0
ZeroDivisionErrorに対応しました。

In [12]: runfile('C:/Python/sample07.py', wdir='C:/Python')

0か、1を入力:1
3.0
NameErrorに対応しました。

In [13]
```

0を入力するとZeroDivisionErrorになる

1を入力すると3になる（除算記号が / なので、小数点以下も含む）

print(test)でエラーになるため、print('正常終了')は実行されない

講師 かなり駆け足でPythonの基本を学習してきたけど、どうでしたか? どうやら、秋にも今回の研修の続きがあるようなので、それぞれの部署に戻っても、Pythonのプログラミングを続けてくださいね。お疲れさまでした。

進 **愛** ありがとうございました。お疲れさまでした。

■筆者紹介

中島 省吾

有限会社メディアプラネット代表取締役。テクニカルライターとして、ネットワークやプログラミング関連の記事を執筆するほか、IT企業向けのセミナーや新人研修の講師なども手掛ける。最近は、ボランティアで子どもたちにもプログラミングを教えている。近著に「ビジネスPython超入門」（日経BP）がある。

■表紙イラスト　ぷちめい

Pythonが5日でわかる本 基本編

発行人●村上 広樹
編集長●久保田 浩
発行●日経BP
Nikkei Business Publications, Inc.
発売●日経BPマーケティング
〒105-8308 東京都港区虎ノ門4-3-12
URL●https://nkbp.jp/bpshopqa

日経BP