

Pythonの基礎 1

Python Basics 1

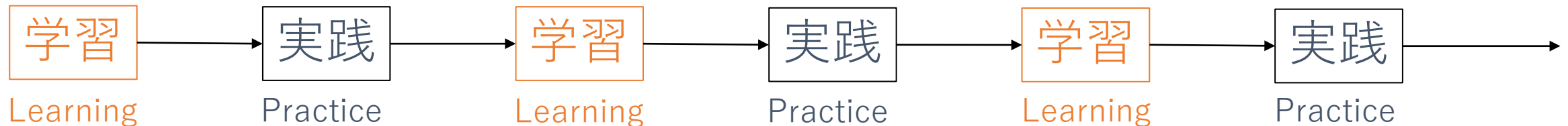


全体の流れ Overall flow

1. Pythonに触れる
2. Pythonの基本
3. 条件分岐と繰り返し
4. ユーザー定義関数
5. 信号処理
6. その他の応用

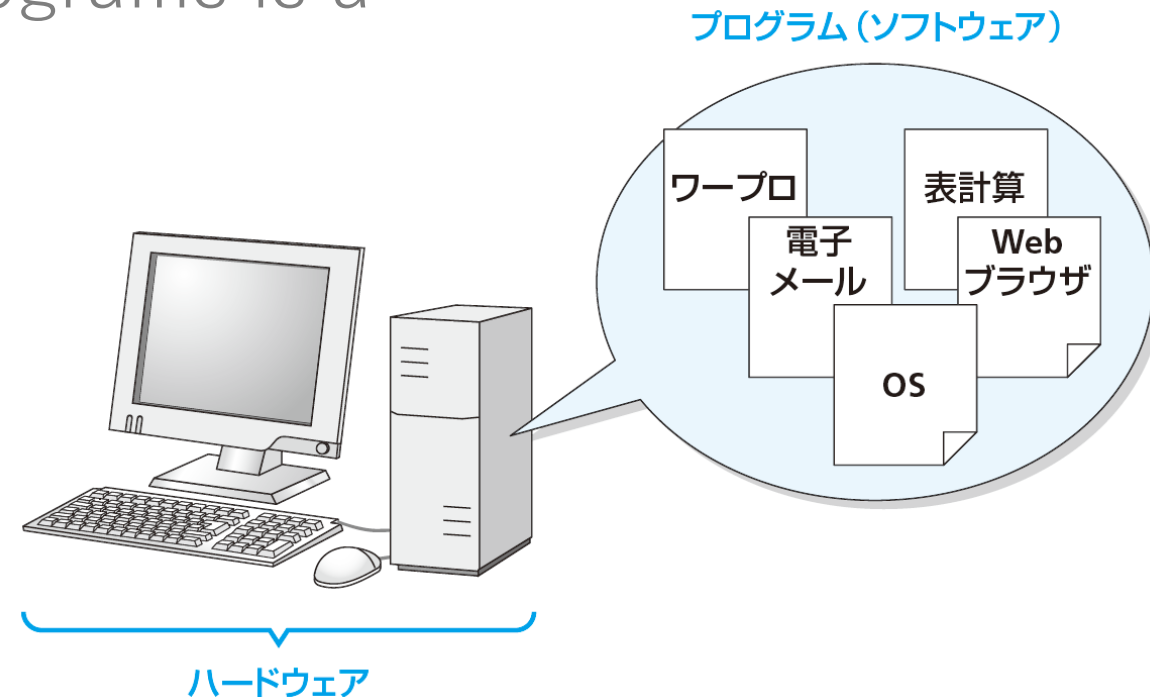
1. touch on Python
2. the basics of Python
3. conditional branching and repetition
4. user-defined functions
5. signal processing
6. other applications

- ▶ 実際に手を動かしてプログラムコードを入力する、一部を変更して試す。
- ▶ Webで公開されているプログラムコードを動かしてみる、一部を変更して試す。
- ▶ Try to enter the programme code by actually working with your hands, change some of it and try it out.
- ▶ Run the programme code published on the web, change some of it and try it out.



プログラムとは What is a computer program?

- ▶ コンピュータに命令を与えるものが「**プログラム**」
- ▶ プログラムを作成するための専用言語が「**プログラミング言語**」
- ▶ その中の1つに「**Python**」がある
- ▶ A '**program**' is what gives instructions to a computer
- ▶ A dedicated language for creating programs is a "**programming language**"
- ▶ One of these languages is '**Python**'

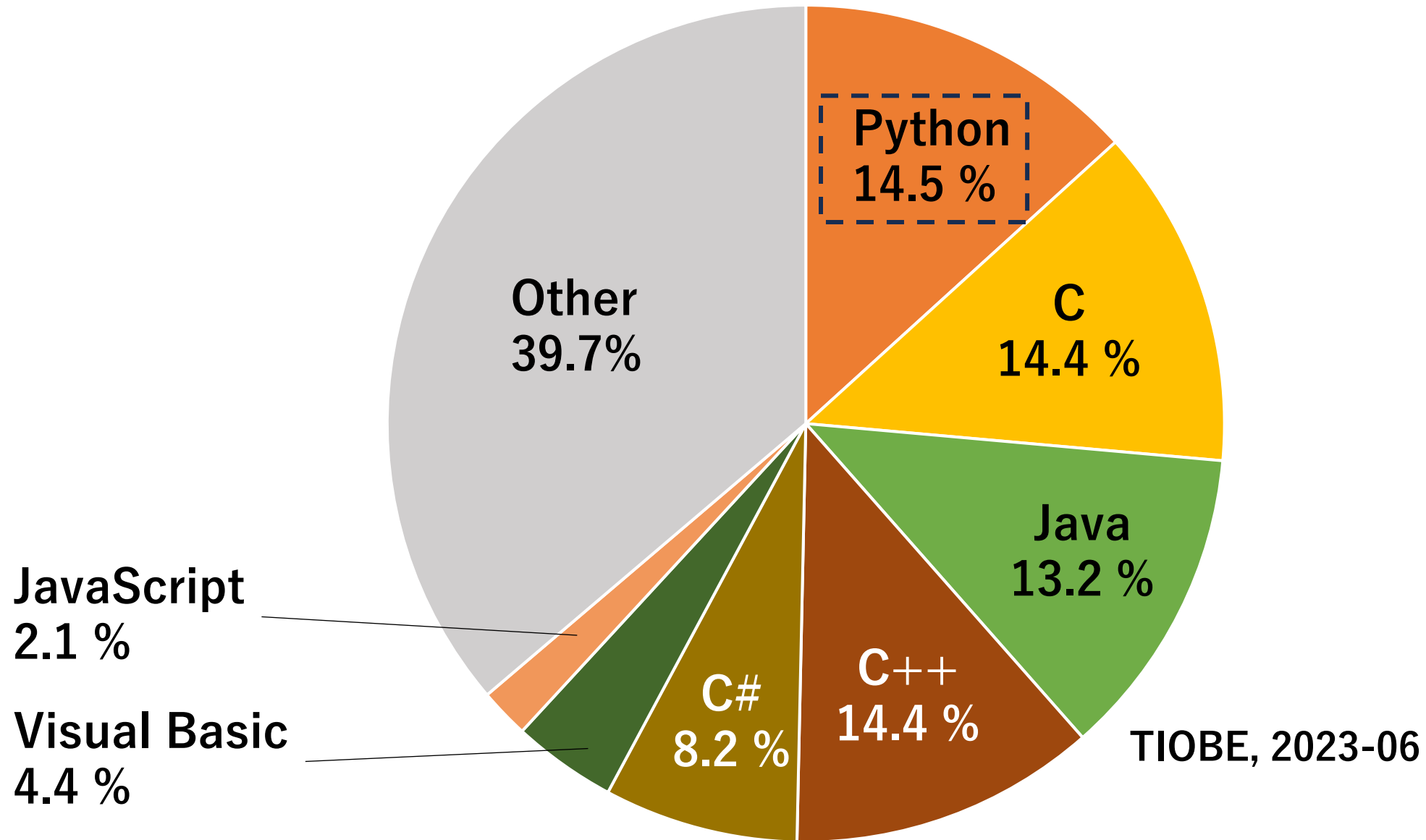


さまざまなプログラミング言語 Programming languages

C	歴史のある言語、組み込みプログラム Language with a long history, embedded programming
C++	C言語の後継、オブジェクト指向 Successor to C language, object-oriented
C#	C++言語の後継、米マイクロソフト Successor to C++ language, US Microsoft
Perl	スクリプト言語、手軽な開発 Scripting language, easy development
PHP	サーバサイド、Webページ生成 Server-side, web page generation
Java	オブジェクト指向、大規模システム Object-oriented, large-scale systems
JavaScript	ブラウザで動作、動的なWebページ Runs in browser, dynamic web pages
Python	修得が容易、機械学習分野で普及 Easy to learn, popular in machine learning
R	統計解析向けの言語 Language for statistical analysis

プログラミング言語のトレンド

Trend of Programming languages



プログラミング言語ランキング（ブラウザ検索数による）

Programming language ranking (by number of browser searches)

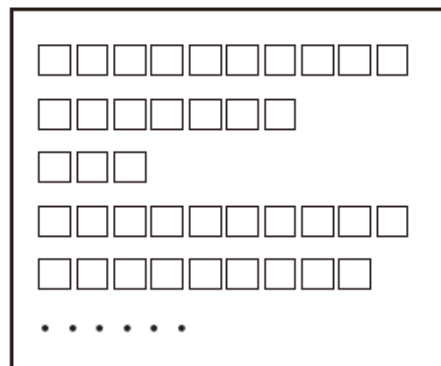
プログラムコードが実行されるまで

Execution of program code

コンパイラ方式

Compiler method

プログラムコード



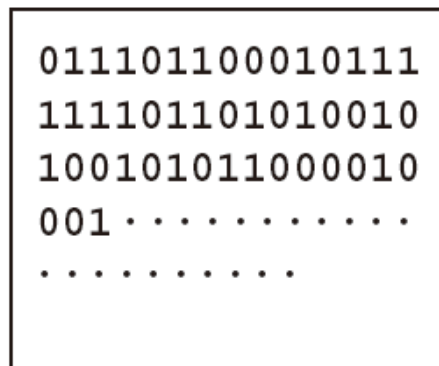
Program code

コンパイル

①



機械語



Machine code

②

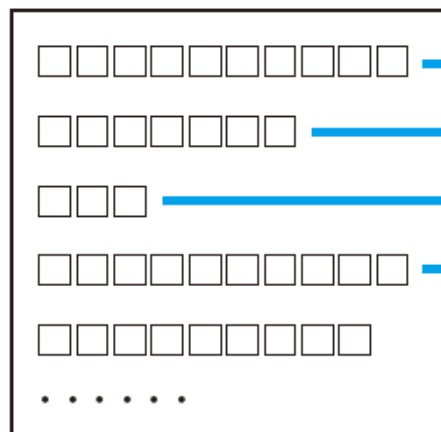


- ▶ 実行速度が高速
- ▶ 実行に手間がかかる
- ▶ デバッグがしづらい
- ▶ Fast execution speed
- ▶ Time-consuming to run
- ▶ Difficult to debug

インタプリタ方式

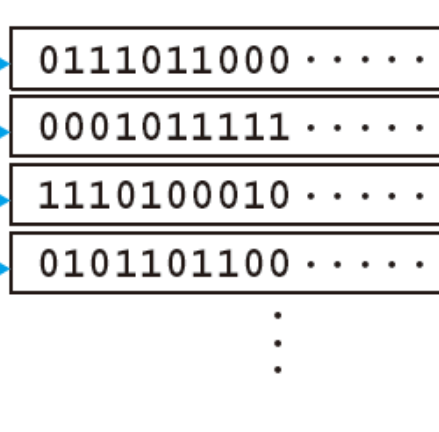
Interpreter method

プログラムコード



Program code

機械語



Machine code

①

③

⑤

⑦

⋮

②

④

⑥

⑧

⋮



- ▶ 実行速度が低速
- ▶ すぐに実行できる
- ▶ デバッグがしやすい
- ▶ Slow execution speed
- ▶ Quick to run
- ▶ Easy to debug

Pythonはインタプリタ方式
Python is an interpreter method

1. Pythonに触れる

touch on Python



Pythonのプログラムコード Program code of python

```
total = 0
for i in range(1, 101):
    total += i
print(total)
```

↑ 1から100までの整数を順番に足して、その結果を画面に表示するプログラムコード

↑ Program code to add integers from 1 to 100 in sequence and display the result on the screen.

- ▶ 半角英数と記号で記述する（文字列は日本語可）
- ▶ 上から順に一行ずつ実行
- ▶ インデント（空白）に意味がある
- ▶ コメントアウトした箇所は、処理に影響しない →
- ▶ Use single-byte alphanumeric characters and symbols
- ▶ Execute one line at a time, starting from the top
- ▶ Indentation (whitespace) is meaningful
- ▶ Commented out sections do not affect processing →

この行はコメントアウト

"""

この部分も
複数行のコメントとなり
プログラムに影響しない

"""

Comment out this line

"""

This part is also It is a
multi-line comment and
and has no effect on the
program.

"""

Pythonのプログラムコード Program code of python

```
total = 0
for i in range(1, 101):
    total += i
print(total)
```

↑ 1から100までの整数を順番に足して、その結果を画面に表示するプログラムコード

↑ Program code to add integers from 1 to 100 in sequence and display the result on the screen.

- ▶ 大文字と小文字は区別される ▶ case sensitive

○ `print('Hello')`

× `Print('Hello')`

- ▶ 単語や数字、記号の前後には、半角の空白文字を入れても入れなくてもよい

- ▶ Words, numbers and symbols may or may not be preceded or followed by a single-byte space character

○ `3+4`

○ `3 + 4`

実行方法 Implementation method

1. 対話モード（インタラクティブモード） Interactive mode

プログラムコードを1行ずつ与え、その都度実行する

Give one line of programme code at a time and run it each time.

2. スクリプトモード Script mode

プログラムコードを記述したファイルを作成し、そのファイルを与えて実行させる

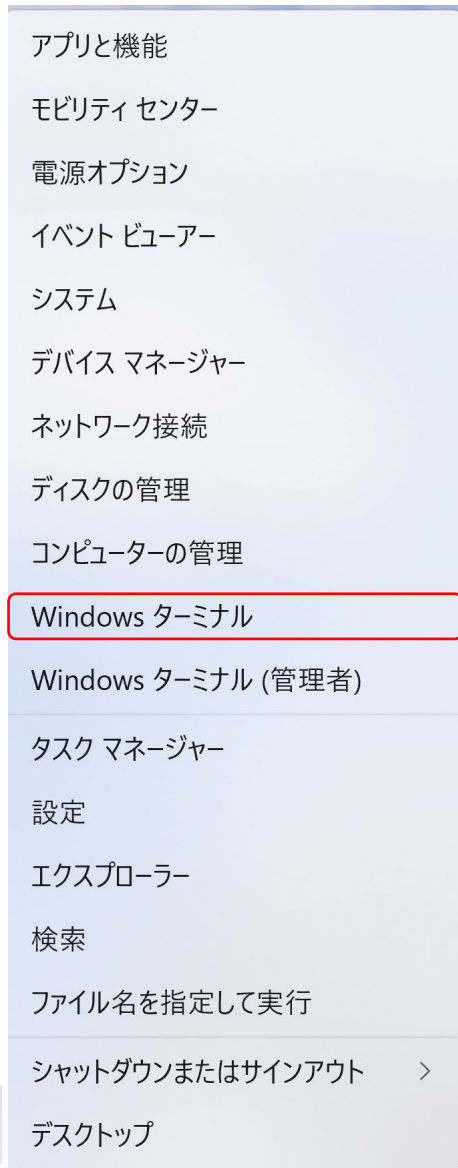
Create a file describing the program code and give it to the programmer to run.

➡ 対話モードでPythonを動かしてみよう

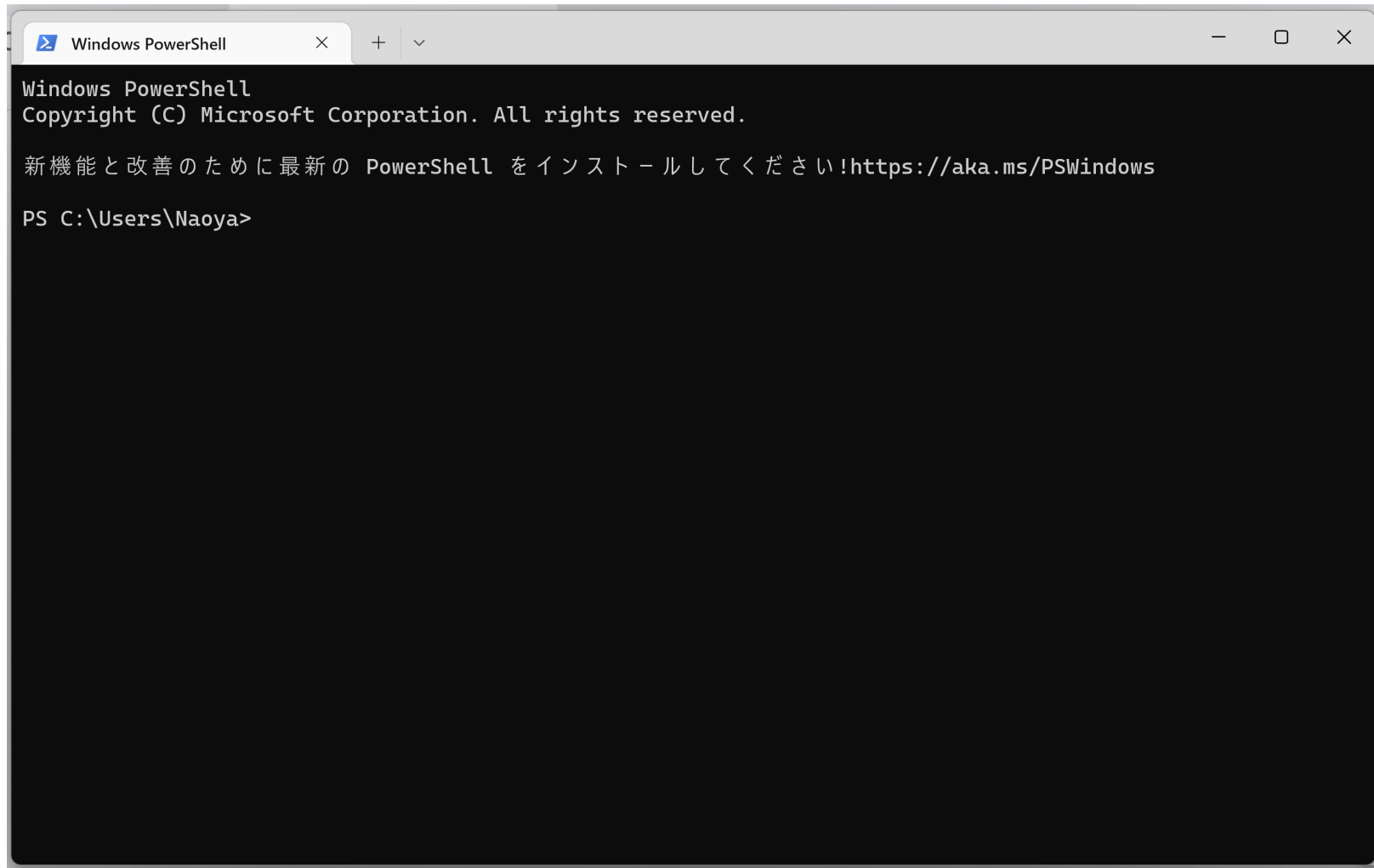
Let's run Python in interactive mode !

Windows PowerShell の起動

Starting Windows PowerShell



Right-click
右クリック



Macの人はターミナル

“Terminal” on Mac

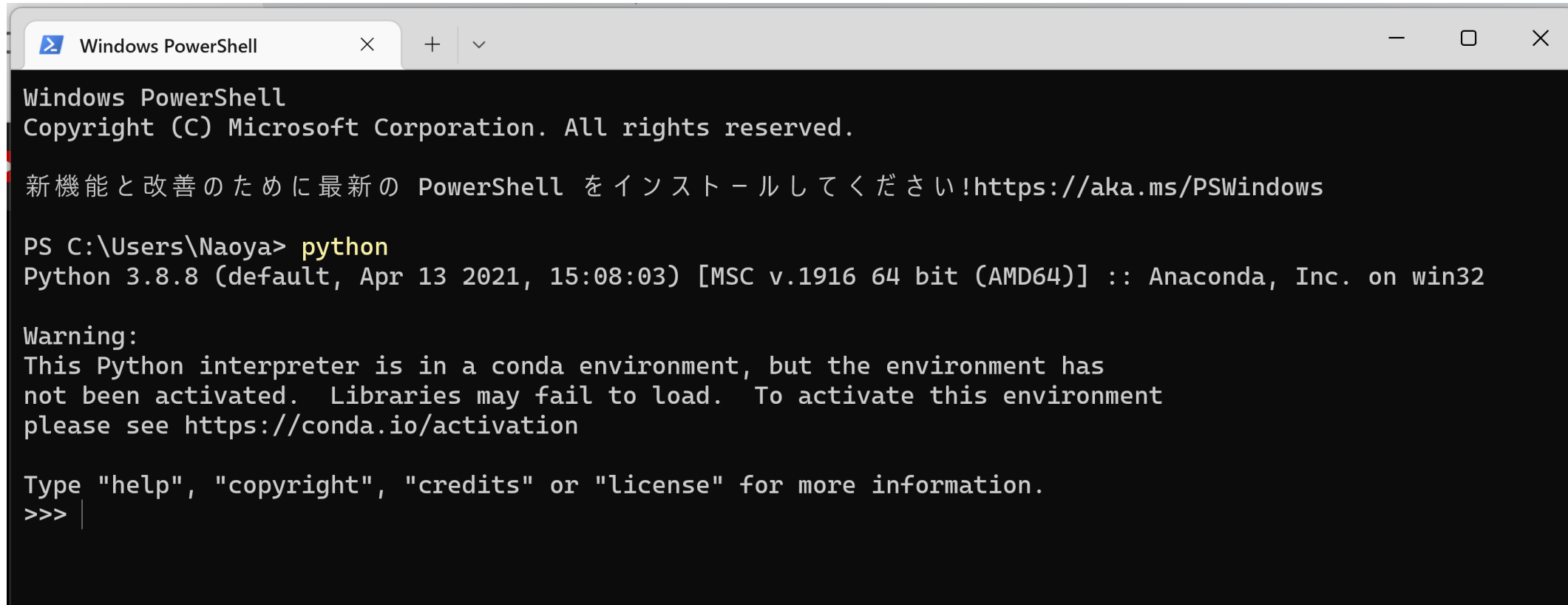
Anything that can run python
in interactive mode.

Windows PowerShell の起動

Starting Windows PowerShell

シェルで「python」と入力

Type “python” in a shell



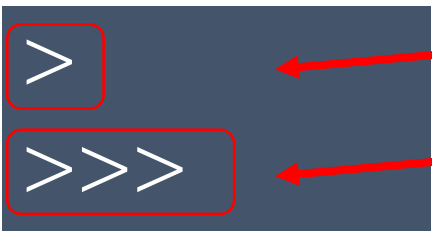
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

新機能と改善のために最新の PowerShell をインストールしてください!https://aka.ms/PSWindows

PS C:\Users\Naoya> python
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license" for more information.
>>> |
```



Windows PowerShell のプロンプト

Windows PowerShell prompts

対話モードのPythonのプロンプト

Python prompt in interactive mode

※ プロンプト：命令を待っている状態を示す記号

Prompt: symbol indicating waiting for instructions

対話モードを電卓のように使ってみる

Use the interactive mode like a calculator.

インタラクティブシェル interactive shell

プロンプト
prompt

>>>

12 + 34

最後に **Enter** キーを押します

46

「コンソール」に結果が出力される

Results are output to the 'console'.

掛け算

multiplication

>>> 15 * 10

150

割り算

division

>>> 90 / 2

45.0

括弧を使った計算

Calculations using
brackets

>>> 10 * (7 - 2) + 5

55

いろいろ試してみよう！

Try things out !

文字列を扱う Strings

```
>>> print('Hello, Python.')
```

Hello, Python.

print (“出力する内容”) として、文字列を出力できる

Can output strings as **print** ("What to output").

文字列の前後をシングルクォーテーション(')、または、ダブルクォーテーション(")で囲む

Single quotation marks (') before and after the string, or, Double quotation marks (") before and after the string.

いろいろ試してみよう！

Try things out !

print関数 print function

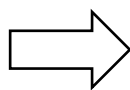
```
print ('Hello')
```

関数

function

引数

arguments



```
>>> print('Hello')  
Hello
```

標準出力

standard output

print 関数は、引数で与えられたものを標準出力に出力する働きをする**組み込み関数**

The **print function** is a built-in function that prints what is given as an argument to standard output.

※ print 関数のように、はじめから使える関数を**組み込み関数**という
(自分で作った関数はユーザ定義関数)

Functions that can be used from the outset, such as the print function, are called **built-in functions**.
(Functions you create yourself are user-defined functions.)

※ 標準出力はプログラム実行環境によって異なる

Standard output differs depending on the program execution environment.

エラー error

```
>>> Print('Hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
```

```
>>> print('Hello)
File "<stdin>", line 1
    print('Hello)
                  ^
SyntaxError: EOL while scanning string literal
```

プログラムコードを適切に実行できない場合に**エラー**が発生し、**エラーメッセージ**が表示される。

An **error** occurs when the program code cannot be executed properly and an **error message** is displayed.

※ エラーメッセージには、発生したエラーに関する説明が表示されるので、がんばって読むようにする。

The error message will contain an explanation of the error that has occurred, Do your best to read it.

変数

variable



変数 variable

「**変数**」とは、値を入れておく入れ物 A '**variable**' is a container in which a value is stored.

a = 3

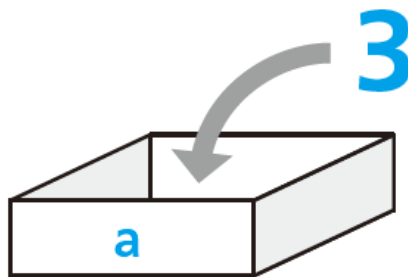
変数名

variable name

← 「変数aに3を**代入**する」 'Assign 3 to variable a'

「代入」の一般的なイメージ

Common images of "assignment"



a という名前のついた箱に 3 を入れる

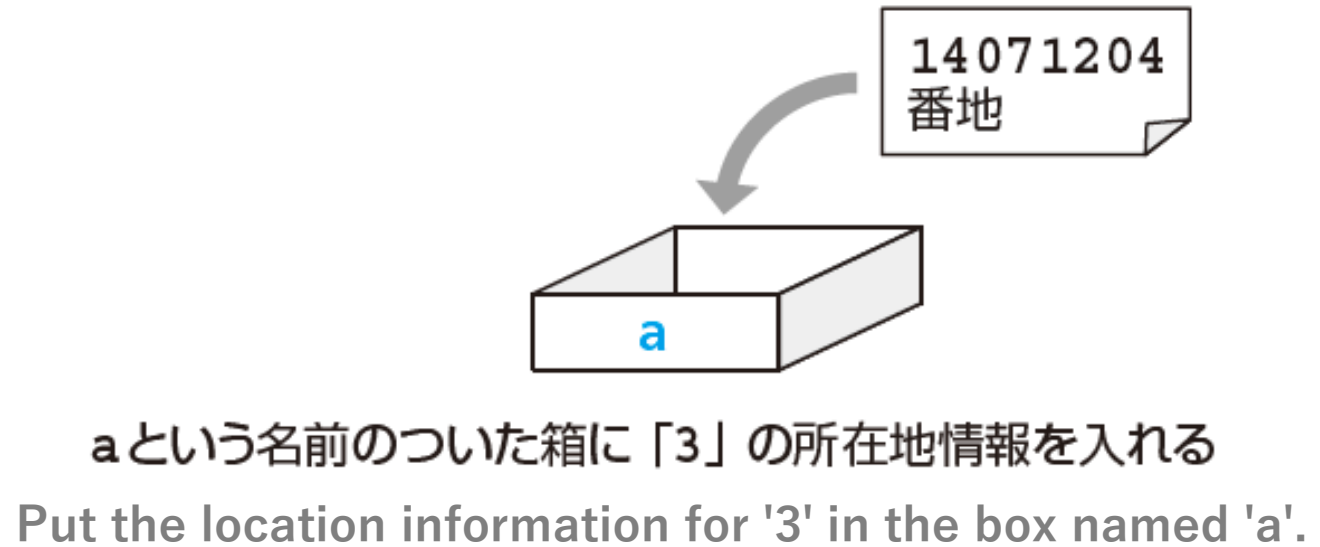
Put 3 into the box named 'a'.

代入の正確なイメージ

Accurate image of the assignment

`a = 3`

←「変数aに3を**代入**する」 “Assign 3 to variable a”



「3という値を表すオブジェクトがコンピュータのメモリ上のどこかに保管される。
その保管場所を示す所在地情報が、aという名前の箱に入れられる」

An object representing the value 3 is stored somewhere in the computer's memory.
The location information for that storage is placed in the box named a.

代入した値を確認する `checking assigned values`

いろいろ試してみよう！

`print` 関数で、変数に代入されている値を出力できる。

Try things out !

The `print` function can be used to output the value assigned to a variable.

```
>>> a = 3  
>>> print(a)
```

← 変数 `a` に 3 を代入します Assign 3 to variable `a`.
← 変数 `a` に代入されている値を出力します Output the value assigned to variable `a`.
3

複数の変数の値をいっぺんに出力できる

The values of several variables can be output at once.

```
>>> a = 10  
>>> b = 123  
>>> print(a, b)
```

← `print` 関数に 2 つの変数を渡しています Two variables are passed to the `print` function.
10 123 ← それぞれの変数の値が出力されました The value of each variable is output.

インタラクティブシェルでは `print` 関数を省略できる

The `print` function can be omitted in interactive shells.

```
>>> a = 3  
>>> a
```

← `print()` を使わずに、変数名だけを記述しています Only the names of the variables are written without `print()`.
3 ← 変数の値が出力されます The value of the variable is output.

代入した値を変更する changing assigned values

変数に、あとから別の値を代入できる

Variables can be assigned different values later.

```
>>> a = 3      ← 変数aに3を代入します    Assign 3 to variable a.
>>> print(a)
3
>>> a = 5      ← 変数aに5を代入します    Assign 5 to variable a.
>>> print(a)
5
>>> a = 'hello' ← 変数aに文字列'hello'を代入します
>>> print(a)    Assign the string "hello" to variable a.
hello
```

いろいろ試してみよう！

Try things out !

練習問題

exercises



問題 1

Question 1

次の文章のうち正しいものには○を、正しくないものには×をつけてください。

○ the correct and × the incorrect in the following sentences.

- (1) コンピュータは、Pythonのプログラムコードを直接理解して処理を行う。

Computers understand and process Python program code directly.

- (2) Pythonのプログラムコードは、大文字と小文字の違いを区別しない。

Python program code is case-insensitive.

- (3) Pythonには、1行ずつプログラムコードを入力して、そのつど実行する方法がある。

Python has a method for entering program code one line at a time and executing it each time.

- (4) 「`print(こんにちは)`」と記述すると、「こんにちは」という文字列が出力される。

"print(hello)" prints the string "hello".

- (5) 変数には後から異なる値を代入できる。

Variables can be assigned different values later.

問題 1 (解答)

Question 1 (Answers)

次の文章のうち正しいものには○を、正しくないものには×をつけてください。

○ the correct and × the incorrect in the following sentences.

× (1) コンピュータは、Pythonのプログラムコードを直接理解して処理を行う。

Computers understand and process Python program code directly.

× (2) Pythonのプログラムコードは、大文字と小文字の違いを区別しない。

Python program code is case-insensitive.

○ (3) Pythonには、1行ずつプログラムコードを入力して、そのつど実行する方法がある。

Python has a method for entering program code one line at a time and executing it each time.

× (4) 「print(こんにちは)」と記述すると、「こんにちは」という文字列が出力される。

"print(hello)" prints the string "hello".

○ (5) 変数には後から異なる値を代入できる。

Variables can be assigned different values later.

問題2 Question 2

対話モードで、次の計算を実行して結果を確認しましょう。

In interactive mode, perform the following calculations to see the results.

(1) $1 + 2 + 3 + 4$

(2) $2 + 3 * 2$

(3) $(2 + 3) * 2$

(4) $10 / 2.5$

(5) $3 / 0$

問題2 (解答)

Question 2 (Answers)

対話モードで、次の計算を実行して結果を確認しましょう。

In interactive mode, perform the following calculations to see the results.

(1) $1 + 2 + 3 + 4$

(2) $2 + 3 * 2$

(3) $(2 + 3) * 2$

(4) $10 / 2.5$

(5) $3 / 0$

```
>>> 1+2+3+4
10
>>> 2+3*2
8
>>> (2+3)*2
10
>>> 10/2.5
4.0
>>> 3/0
ZeroDivisionError: division by zero
>>>
```

2. Pythonの基本

Python basics



型と算術演算

Type and
arithmetic operations



型 type

データの種類のことを「**型**」とよぶ

The kind of data is called a "**type**"

型 type	型名 type name		例 example
int	整数型	integer	-1,0,1,2,10,100
float	浮動小数点型	floating point	小数点を含む値 value including decimal point -0.12, 0.0, 0.5, 2.34
str	文字列型	string	‘hello’, ‘こんにちは’
bool	真偽値型	boolean	True, False

type関数による型の確認

Checking types with the type function

いろいろ試してみよう！

Try things out !

type(値)もしくはtype(変数名)で型を確認できる。

You can check the type by type (value) or type (variable name).

```
>>> type(10)
<class 'int'>
```

← 10はint型であることがわかります 10 is int-type

```
>>> type(0.5)
<class 'float'>
```

← 0.5はfloat型であることがわかります 0.5 is float-type

```
>>> type('こんにちは')
<class 'str'>
```

← 'こんにちは'はstr型であることがわかります 'こんにちは' is str-type

```
>>> type(True)
<class 'bool'>
```

← Trueはbool型であることがわかります True is bool-type

```
>>> a = 10
```

← 変数aに10を代入します Assign 10 to variable a.

```
>>> type(a)
<class 'int'>
```

← aの値 (注②-5) はint型であることがわかります a is int-type

```
>>> b = 'こんにちは'
```

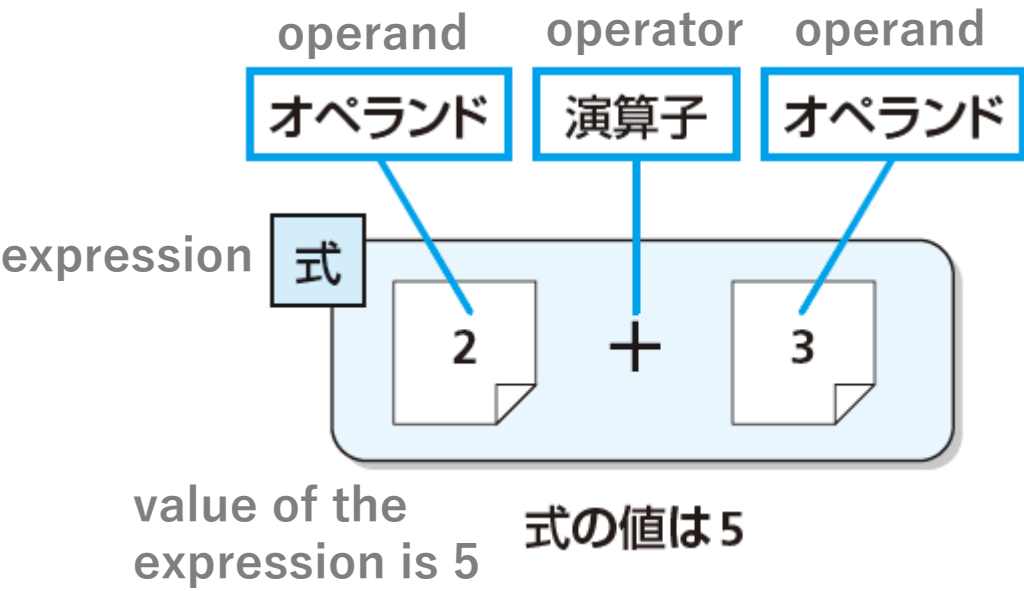
← 変数bに'こんにちは'という文字列を代入します Assign 'こんにちは' to variable b.

```
>>> type(b)
<class 'str'>
```

← bの値はstr型であることがわかります b is str-type

算術演算 arithmetic operations

覚えるべき用語 Terms to remember



文 (代入文) statement

```
a = 2 + 3
```

リテラル literal
(プログラムコード中に記述される数値)
Numerical values described in the program code

演算子の種類 Types of perators

演算子 operator	演算の内容 Description of operation	
+	加算 (足し算)	addition
-	減算 (引き算)	subtraction
*	乗算 (掛け算)	multiplication
/	除算 (割り算)	division
//	商	quotient
%	剰余	remainder
**	べき乗	power
:=	代入	substitution

変数を含む算術演算 Arithmetic operations involving variables

式に変数名が含まれる場合は、
変数に代入されている値が使用される

If the expression contains a variable name,
the value assigned to the variable is used.

```
>>> a = 10  
>>> print(a + 3)  
13
```

← 変数aに10を代入します Assign 10 to variable a.

← 式 a + 3の値を出力します Outputs the value of equation A+3.

```
b = a + 3
```

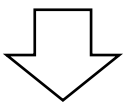
← (変数aの値)+3 が変数bに代入される
(the value of variable a) + 3 is assigned to variable b

```
a = a + 3
```

← (変数aの値)+3 が変数aに代入される
つまりaの値が3増える
(the value of variable a) + 3 is assigned to variable a
In other words, the value of a is increased by 3.

算術演算の短縮表現 Shortened representation of arithmetic operations

a = a + 3



短縮
shortening

a += 3



加算代入
addition and substitution

operator	example	examination
演算子	使用例	説明
+=	a += b	a = a + b と同じ same as a=a+b
-=	a -= b	a = a - b と同じ
*=	a *= b	a = a * b と同じ
/=	a /= b	a = a / b と同じ
%=	a %= b	a = a % b と同じ
//=	a //= b	a = a // b と同じ
**=	a **= b	a = a ** b と同じ

いろいろ試してみよう！

Try things out !

数値の型 numeric type

プログラムコードへの記述のしかたで、型が異なる。

The type differs depending on how it is written in the program code.

- ・ 小数点を含まない → int型
- ・ 小数点を含む → float型

Not including a decimal point → int-type

Including a decimal point → float-type

```
>>> type(0)
```

```
<class 'int'>
```

← 0という表記はint型になります

The notation “0” is an int-type

```
>>> type(0.0)
```

```
<class 'float'>
```

← 0.0という表記はfloat型になります

The notation “0.0” is an float-type

数値の型変換

Numeric Type Conversion

▸ int型 → float型

```
a = float(100)
```

変数aは float 型になる

Variable a becomes a float-type.

▸ float型 → int型

```
a = int(9.6)
```

変数aは int 型になる
小数点以下は切り捨て

Variable a becomes a int-type.

Decimals are rounded down to the nearest whole number.

文字列とリスト

Strings and Lists

パイソン

文字列の扱い string handling

- ▶ +演算子による文字列の連結 Concatenating Strings with the + operator

```
a = 'AAA' + 'BBB'
```

'AAABBB'

```
a = 'AAA'  
b = 'BBB'  
c = a + b
```

'AAABBB'

- ▶ *演算子による連結のくりかえし Repetition of concatenation by * operator

```
a = 'ABC' * 3
```

'ABCAABCABC'

いろいろ試してみよう！

Try things out !

数値→文字列の変換 Numeric to String Conversion

数値を文字列のようには扱えない Numeric values cannot be treated like strings

✗ `a = 500 + '円'`

↓ `str(数値)`で文字列に変換する Convert to string with `str(numeric value)`

○ `a = str(500) + '円'`

```
>>> year = 2021
>>> print(str(year) + '年')
2021年
```

← `year`の値 (int型) を文字列に変換してから連結しています

The value of `year` (type int) is converted to a string and then concatenated.

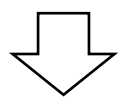
いろいろ試してみよう！

Try things out !

変数の値の埋め込み Embedding variable values

数値を文字列に変換してから連結 Convert numeric values to strings and then concatenate them

```
>>> price = 550
>>> print('この商品は' + str(price) + '円です')
この商品は550円です This item is 550 yen
```



フォーマット文字列を使用して
簡潔に記述できる

Can be described concisely using format strings.

```
>>> price = 550
>>> print(f'この商品は{price}円です')
この商品は550円です
```

f'文字列' とすると、文字列に含まれる {変数名} 部分が変数の値に置き換わる

f'string', the {variable name} part in the string is replaced by the value of the variable

いろいろ試してみよう！

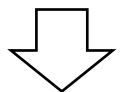
Try things out !

フォーマット文字列の活用 Utilizing Format Strings

フォーマット文字列 format strings

f'文字列' とすると、文字列に含まれる {変数名} 部分が変数の値に置き換わる

f'string', the {variable name} part in the string is replaced by the value of the variable



{変数名} 部分に式を入れることもできる You can also put **expressions** in the {variable name} part.

```
a = 5
b = 550
print(f'1つ{a}円です。{b}個で{a * b}円です')
```

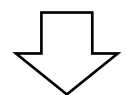
いろいろ試してみよう！

Try things out !

文字列→数値の変換 String to Numeric Conversion

文字列を数値のように扱えない Strings cannot be treated like numbers.

```
a = '500'    ←文字列 string  
b = a * 2    '500500'
```



int(文字列)で整数に変換する

Convert to integer with int (string)

```
a = '500'  
b = int(a) * 2    1000
```

小数点を含む数値に変換するときはfloat(文字列)

float(string) when converting to a number with a decimal point

いろいろ試してみよう！

Try things out !

リスト list

リストを使って、複数の値をまとめて管理できる Lists can be used to manage multiple values at once.

```
a = [10, 20, 30, 40, 50]
```

リストに格納された**要素**には、**インデックス**を使ってアクセスできる

Elements stored in the list can be accessed using indexes

```
>>> print(a[0])  
10  
>>> print(a[1])  
20  
>>> print(a[4])  
50
```

※ インデックスは0から始まる

いろいろ試してみよう！

Try things out !

リスト list

リストを使って、複数の値をまとめて管理できる Lists can be used to manage multiple values at once.

```
a = [10, 20, 30, 40, 50]
```

インデックスにはマイナスの値も使える Negative values can be used for indexes.

```
>>> print(a[-1]) ← 末尾の要素
50
>>> print(a[-2]) ← 後ろから2番目の要素
40
>>> print(a[-5]) ← 後ろから5番目の要素
10
```

いろいろ試してみよう！

Try things out !

リストのスライス（範囲指定） List slicing (range specification)

```
a = [10, 20, 30, 40, 50]
```

```
>>> print(a[0:2])  ← 0番目から1番目までの要素 0th to 1st element
[10, 20]
>>> print(a[:3])  ← 0番目から2番目までの要素（省略→0） 0th to 2nd element (omitted→0)
[10, 20, 30]
>>> print(a[3:])  ← 3番目から最後まで 3rd to last.
[40, 50]
>>> print(a[::2])  ← 0番目から最後まで2個ずつ 2 each from the 0th to the last.
[10, 30, 50]
>>> print(a[::-1]) ← 0番目から最後までを逆順に取得
[50, 40, 30, 20, 10] Obtain from 0th to last in reverse order
```

```
a[A:B:C]
```

リストaのA~(B-1)までの要素をC個ずつ
C elements from A~(B-1) in list a by C

いろいろ試してみよう！

Try things out !

リスト内の値の変更 Changing values in a list

インデックスを指定して値を変更できる Can change values by specifying **indexes**

```
>>> a = [10, 20, 30, 40, 50]
>>> print(a)
[10, 20, 30, 40, 50]
```

← リストの全要素を出力
Output all elements of the list

```
>>> a[0] = 99
>>> print(a)
[99, 20, 30, 40, 50]
```

← 先頭の要素を99に変更
Change the first element to 99

```
>>> a[-1] = 'A'
>>> print(a)
[99, 20, 30, 40, 'A']
```

← 文字列にもできる
It can be a string.

いろいろ試してみよう！

Try things out !

リストの操作 List operations

追加(append)・削除(remove)・並び替え(sort)

```
>>> a = [10, 40, 30, 20, 50]
```

```
>>> a.append(100)
```

← 要素100を追加 Add element 100

```
>>> print(a)
```

```
[10, 40, 30, 20, 50, 100]
```

```
>>> a = [10, 40, 30, 20, 50]
```

```
>>> a.remove(20)
```

← 要素20を削除 Remove element 20

```
>>> print(a)
```

```
[10, 40, 30, 50]
```

```
>>> a = [10, 40, 30, 20, 50]
```

```
>>> a.sort()
```

← 昇順に並び替え Sort in ascending order

```
>>> print(a)
```

```
[10, 20, 30, 40, 50]
```

リストの要素数の確認

Checking the number of elements in a list

len関数でリストの要素数を取得できる

The len function can be used to get the number of elements in a list.

```
>>> a = [10, 20, 30, 40, 50]
>>> len(a)
5
```

リストの生成(rangeオブジェクト) Generating a list (range object)

range オブジェクトの生成方法	得られる整数の列 The resulting sequence of integers
<code>range(stop)</code>	0から「stopの値-1」までの整数 Integer from 0 to "stop value -1"
<code>range(start, stop)</code>	startの値から「stopの値-1」までの整数 Integer from "start" value to "stop" value -1
<code>range(start, stop, step)</code>	startの値から「stopの値-1」までの整数。ただし、増分はstepの値
	Integer from the value of start to the value of stop -1, where the increment is the value of step

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(3, 10))
[3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 30, 10))
[1, 11, 21]
```

← 0から9までの数字が並びます Numbers from 0 to 9

← 3から9までの数字が並びます Numbers from 3 to 9

← 29を超えない範囲で1から10ずつ値が増えます

Increments by 10 from 1 within the range not exceeding 29

練習問題2

exercises2



問題 1

Question 1

次の文章のうち正しいものには○を、正しくないものには×をつけてください。

○ the correct and × the incorrect in the following sentences.

- (1) 一度int型の値を代入した変数aに対して、後から文字列を代入することはできない。

Once a variable a is assigned a value of type int, it is not possible to assign a string to it later.

- (2) int型の値とfloat型の値を加算するときには、その前にint型の値をfloat型に型変換しておく必要がある。

When adding a value of type int to a value of type float, the value of type int must first be converted to a value of type float. type conversion to a float type before adding values of type int.

- (3) int型とfloat型の値を含む算術演算の結果はfloat型になる。

The result of an arithmetic operation involving values of type int and float will be of type float.

- (4) `a = int(3.8)`と記述した場合、変数aの値は4になる。

If `a = int(3.8)`, the value of variable a is 4.

問題 1

Question 1

次の文章のうち正しいものには○を、正しくないものには×をつけてください。

○ the correct and × the incorrect in the following sentences.

- × (1) 一度int型の値を代入した変数aに対して、後から文字列を代入することはできない。

Once a variable a is assigned a value of type int, it is not possible to assign a string to it later.

- × (2) int型の値とfloat型の値を加算するときには、その前にint型の値をfloat型に型変換しておく必要がある。

When adding a value of type int to a value of type float, the value of type int must first be converted to a value of type float. type conversion to a float type before adding values of type int.

- (3) int型とfloat型の値を含む算術演算の結果はfloat型になる。

The result of an arithmetic operation involving values of type int and float will be of type float.

- × (4) `a = int(3.8)`と記述した場合、変数aの値は4になる。

If `a = int(3.8)`, the value of variable a is 4.

問題2 Question 2

pythonで次の値を求める計算式を書いてください。

Write an equation to find the following values in python.

(1) 100を9で割った商

The quotient of 100 divided by 9

(2) 1000を7で割った余り

The remainder of 1000 divided by 7

(3) 3の5乗

3 to the fifth power

問題2 Question 2

pythonで次の値を求める計算式を書いてください。

Write an equation to find the following values in python.

(1) 100を9で割った商

The quotient of 100 divided by 9

`100 // 9`

(2) 1000を7で割った余り

The remainder of 1000 divided by 7

`1000 % 7`

(3) 3の5乗

3 to the fifth power

`3 ** 5`

問題3 Question 3

以下のプログラムコードを実行した後の変数aの値を教えてください

Answer the value of variable **a** after executing the following program code

(1) `a = 3`
 `a *= 3`

(2) `b = 2`
 `a = b * b`

(3) `a = int(1.9)`

(4) `x = 'XXX'`
 `y = 'YYY'`
 `a = x + y`

問題3

Question 3

以下のプログラムコードを実行した後の変数aの値を教えてください

Answer the value of variable **a** after executing the following program code

```
(1)  a = 3
      a *= 3
```

```
(2)  b = 2
      a = b * b
```

(3) `a = int(1.9)` 1

```
(4)  x = 'XXX'
      y = 'YYY'
      a = x + y
```

XXXXYYY

問題4 Question 4

「I am 21 years old」という文字列が出力されるように作成した次のプログラムコードは、実行するとエラーが発生します。適切に動作するように修正してください。

The following program code, created to output the string "I am 21 years old," produces an error when executed. Please modify it to work properly.

```
>>> age = 21
>>> print('I am ' + age + ' years old.')
```


問題4 Question 4

「I am 21 years old」という文字列が出力されるように作成した次のプログラムコードは、実行するとエラーが発生します。適切に動作するように修正してください。

The following program code, created to output the string "I am 21 years old," produces an error when executed. Please modify it to work properly.

```
>>> age = 21
>>> print('I am ' + age + ' years old.')
```

```
>>> age = 21
>>> print('I am ' + str(age) + ' years old.')
```

フォーマット文字列を使う場合 using format strings

```
>>> age = 21
>>> print(f'I am {age} years old.')
```

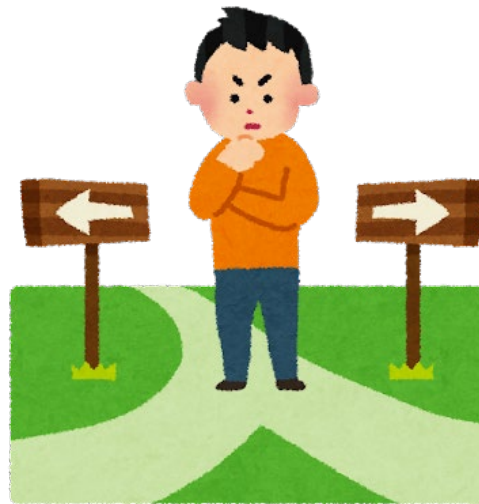
Pythonの基礎 2

Python Basics 2



条件分岐と論理演算子

Conditional branches and
repetition



if文による条件分岐 Conditional branching by if statement

「もしも〇〇ならば××を実行する」 If 〇〇, then run ××

構文 sentence

半角スペース Half-width space

conditional expression

if 条件式 : ← コロンをつける put colon

処理

processing

インデントを空ける leave space for indentation

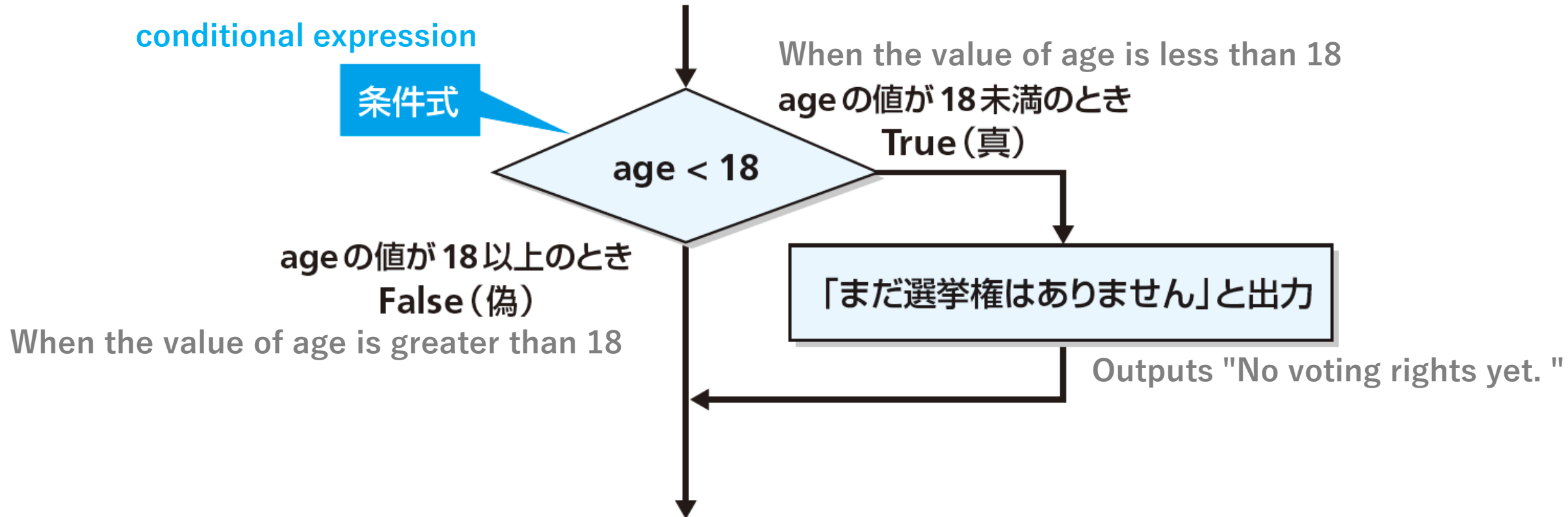
(半角スペース×4 or Tabキー) Half-width space x 4 or Tab key

条件を満たすとき（**条件式**の値が真（**True**）のとき）、処理内容が実行される。
そうでないとき（False）は実行されない。

When the condition is satisfied (the value of the conditional expression is **true**),
the process content is executed.
Otherwise (False), it is not executed.

if文による条件分岐 Conditional branching by if statement

```
age = 16
if age < 18:
    print('まだ選挙権はありません')
    No voting rights yet.
```



if文による条件分岐 Conditional branching by if statement

‘Please tell me your age:’

```
age = int(input('年齢を教えてください: '))  
if age < 18:    No voting rights yet.  
    print('まだ選挙権はありません')  
    print('18歳になったら投票に行きましょう')  
    When you turn 18, you can vote.  
print('処理を終わります') I will finish the process.
```

←
キーボードから入力された数字を
int型にして、変数ageに代入します

The number entered from the
keyboard is converted to an int type
and assigned to the variable age.

input関数 input function

- ▶ 実行時にキーボードで入力したデータを戻り値とする関数
- ▶ 戻り値は必ず**文字列**
- ▶ Function returns data entered on the keyboard at runtime
- ▶ Return value is always a **string**

条件式と関係演算子 Conditional Expressions and Relational Operators

演算子 operators	説明 description	例 example	
==	左辺と右辺が等しい Left and right sides are equal	a == 1	aが1のときにTrue True when a is 1
!=	左辺と右辺が等しくない Left and right sides are not equal	a != 1	aが1でないときにTrue True when a is not 1
>	左辺が右辺より大きい Left side is greater than right side	a > 1	aが1より大きいときにTrue True when a is greater than 1
<	左辺が右辺より小さい Left side is less than right side	a < 1	aが1より小さいときにTrue True when a is less than 1
>=	左辺が右辺より大きい か等しい Left side is greater than or equal to the right side	a >= 1	aが1以上のときにTrue True when a is greater than or equal to 1
<=	左辺が右辺より小さい か等しい Left side is less than or equal to the right side	a <= 1	aが1以下のときにTrue True when a is less than or equal to 1

if~else文による条件分岐 Conditional branching by if~else statement

「もしも〇〇ならば××を実行し、そうでなければ△△を実行する」

If 〇〇, then run ××, otherwise △△

構文 sentence

半角スペース Half-width space

conditional expression

if 条件式:

コロンをつける put colon

処理1 process 1

else:

コロンをつける put colon

処理2 process 2

インデントを空ける leave space for indentation

(半角スペース×4 or Tabキー) Half-width space x 4 or Tab key

条件を満たすとき（**条件式**の値が真（**True**）のとき）、**処理1**が実行される。
そうでないとき（False）は、**処理2**が実行される。

When the condition is satisfied (the value of the conditional expression is true),
process 1 is executed. Otherwise (False), **process 2** is executed.

if~else文による条件分岐 Conditional branching by if~else statement

```
age = 18
```

```
if age < 18:    No voting rights yet.
```

```
    print('まだ選挙権はありません')
```

```
else:
```

```
    print('投票に行きましょう')
```

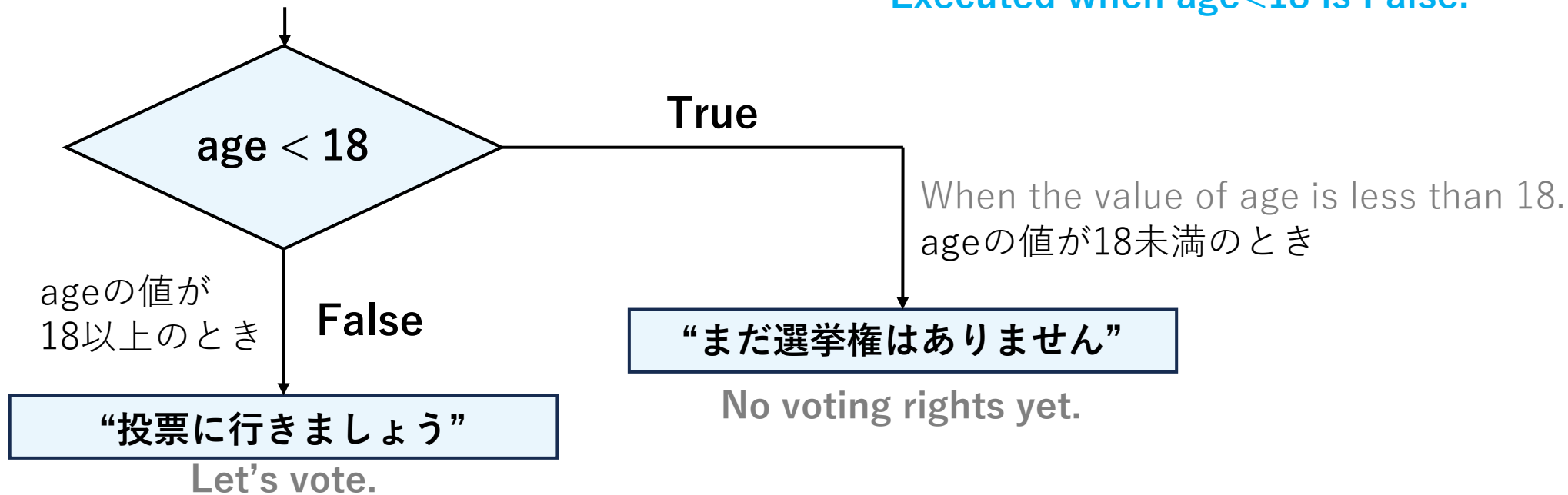
Let's vote.

Executed when age<18 is True.

← age < 18がTrueのときに実行されます

← age < 18がFalseのときに実行されます

Executed when age<18 is False.



if~elif~else文による条件分岐 Conditional branching by if~elif~else statement

構文 sentence

```
if 条件式1: conditional expression1
    処理1 process 1
elif 条件式2: conditional expression2
    処理2 process 2
else:
    処理3
```

条件式1 が True のとき **処理1** が実行される。
そうでないとき、**条件式2** が True のとき **処理2** が実行される。
それ以外の時には、**処理3**が実行される。

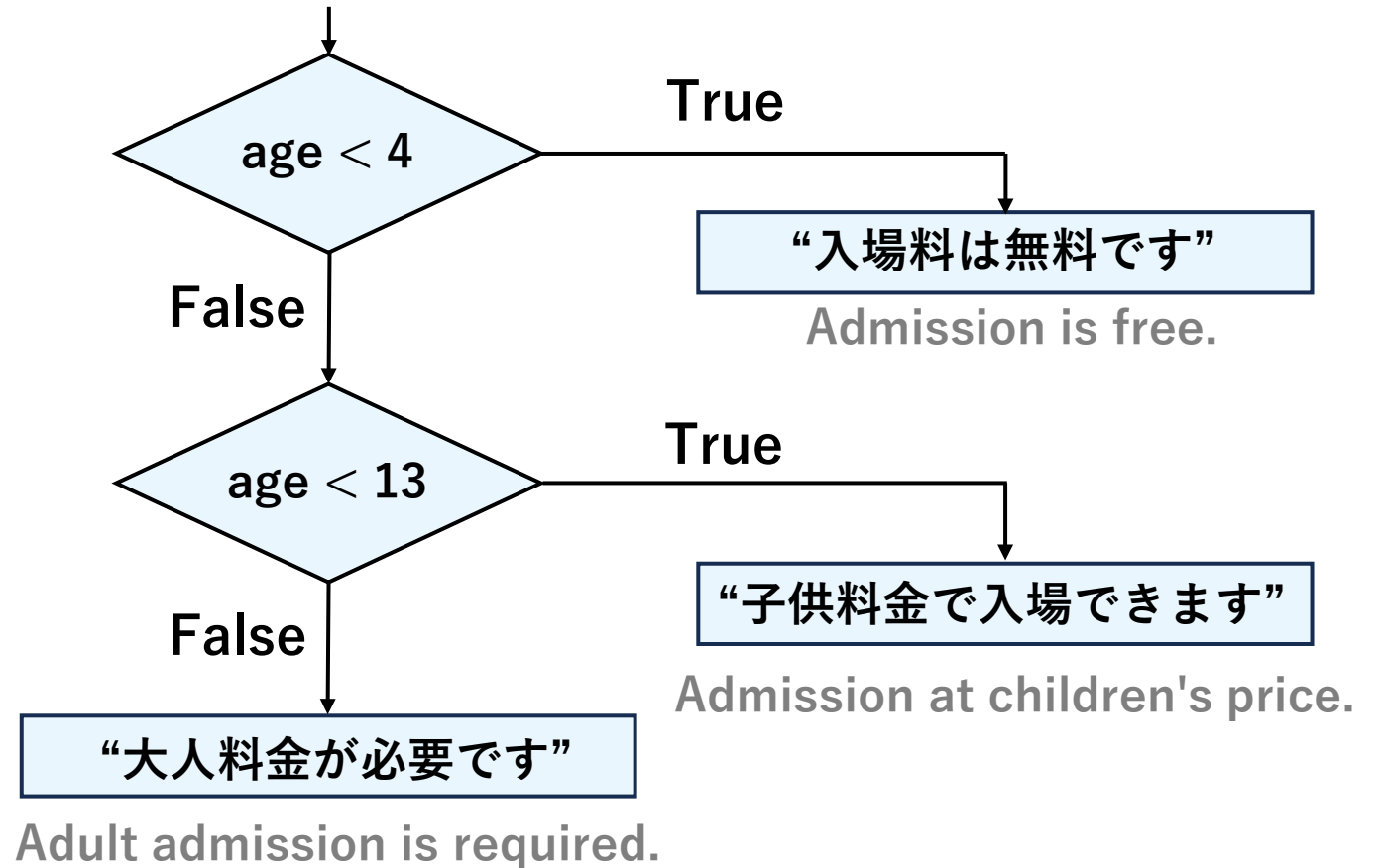
When conditional expression 1 is True, process 1 is executed.

Otherwise, when the conditional expression 2 is True, process 2 is executed.

Otherwise, process 3 is executed.

if~elif~else文による条件分岐 Conditional branching by if~elif~else statement

```
age = 20
if age < 4: Admission is free.
    print('入場料は無料です')
elif age < 13:
    print('子供料金で入場できます')
    Admission at children's price.
else:
    print('大人料金が必要です')
    Adult admission is required.
```



三項演算子 conditional operator

値1 if 条件式 else 値2

Value-1

conditional
expression

Value-2

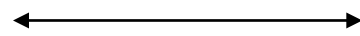
条件式 が **True** のとき、**値1** になる。そうでないとき、**値2** になる。

If the conditional expression is True, the value is Value-1. Otherwise, the value is Value-2.

例

c = a if a > b else b

同じ Same



if a > b:

c = a

else:

c = b

c の値は a になる（もし $a > b$ なら）。
そうでなければ b になる。

The value of c will be a (if $a > b$).
Otherwise, it will be b.

論理演算子による条件の組み合わせ Combining conditions with logical operators

「変数aが10で、**かつ**変数bが5である」

Variable a is 10 **and** variable b is 5



```
a == 10 and b == 5
```

「変数aが10である、**または**変数bが5である」

Variable a is 10 **or** variable b is 5



```
a == 10 or b == 5
```

論理演算子 Logical operators

演算子 operators	動作 operation	式がTrueになる条件 Condition for expression to be True
and	論理積 logical conjunction	左辺と右辺の両方がTrueのとき When both the left and right sides are True
or	論理和 logical disjunction	左辺と右辺の少なくともどちらかがTrueのとき True when a is not 1
not	否定 negation	右辺がFalseのとき True when a is greater than 1

```
age =   
if age < 13 or age >= 65:  
    print("Admission is free")  
else:  
    print("An admission fee is required")
```

ageが13より小さいもしくは65以上の時、入場料が無料になる。
Admission is free when age is less than 13 or more than 65.

演算子の優先度とカッコ

Operator Precedence and Parentheses

a + 10 > b * 5

↕

(a + 10) > (b * 5)

a > 10 and b < 3

↕

(a > 10) and (b < 3)

優先順位 priority	演算子 operators
高い High	**
	* / % //
	+ -
	< > <= >= == !=
	not
	and
	or
低い Low	:=

比較演算子の連結 Comparison operator concatenation

$(a > 5) \text{ and } (a < 10)$



$5 < a < 10$

※ 左から順番に評価される
($5 < a$ が評価されてから $a < 10$ が評価される)

Evaluated from left to right
($5 < a$ is evaluated before $a < 10$ is evaluated)

条件分岐と繰り返し

Conditional branches and
repetition



while文による処理の繰り返し Repetition of processing by while statement

構文 sentence

conditional expression

while 条件式 :

処理

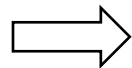
processing

条件式の値がTrueの間、処理を繰り返す。

The process is repeated while the value of the conditional expression is True.

```
i = 0
while i < 5:
    print("hello world")
    i = i + 1
```

実行結果



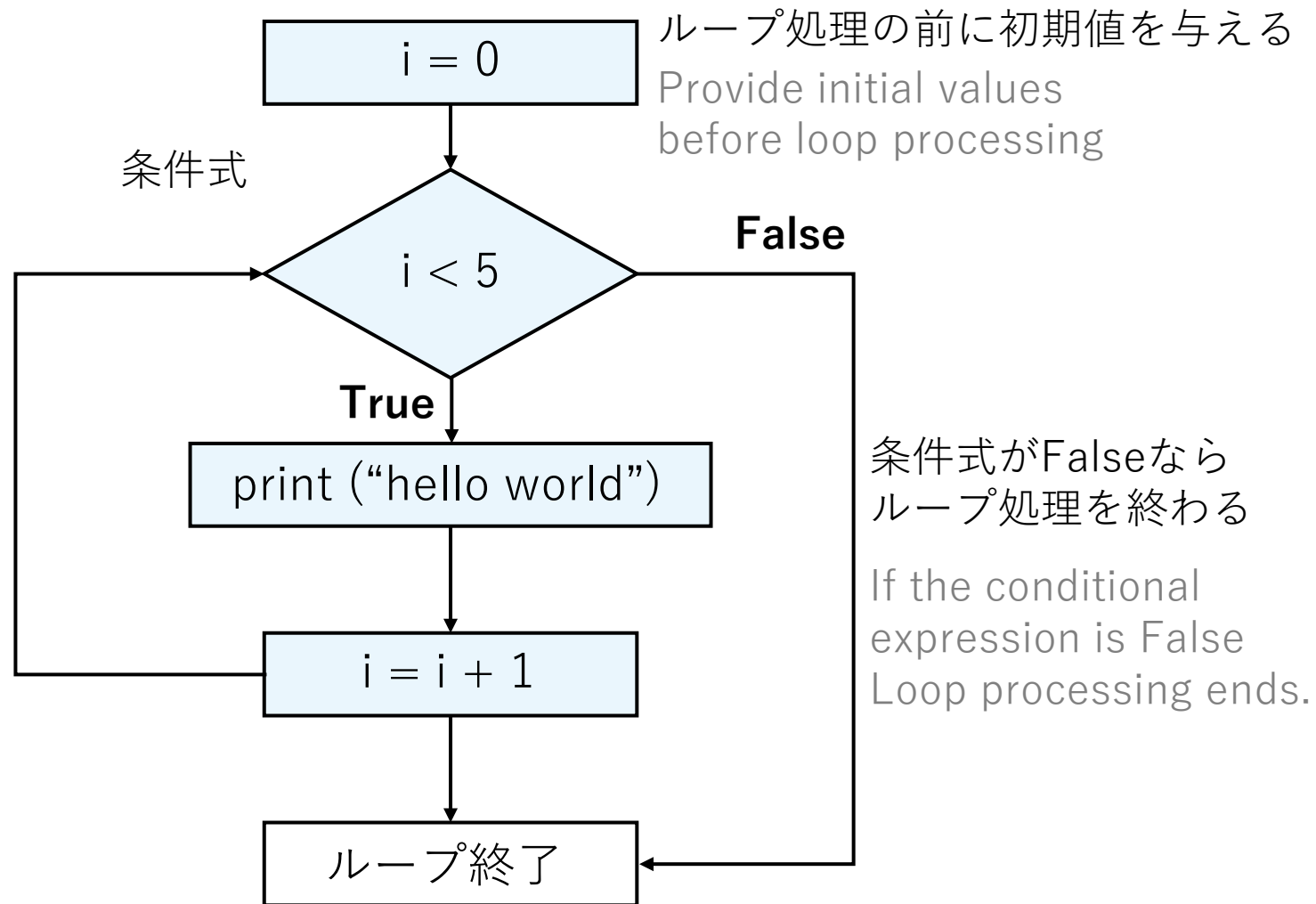
Execution Result

```
hello world
hello world
hello world
hello world
hello world
```

処理の流れ（フローチャート） Flowchart

```
i = 0
while i < 5:
    print("hello world")
    i = i + 1
```

条件式がTrueなら
ループ処理を続ける
If the conditional expression is True
Loop processing continues.



while文の例 example of while statement

```
i = 20
while i > 0:
    print(i)
    i -= 5
```

← 変数iの値が0より大きければ次のブロックの処理を繰り返す
If the value of variable i is greater than 0, repeat the process for the next block.

← 変数iの値を5減らす Decrease the value of variable i by 5.

実行結果 Execution Result

```
20
15
10
5
```

for文による処理の繰り返し

Repetition of processing by for statement

構文 sentence

variable

iterable object

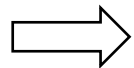
```
for 変数 in 反復可能オブジェクト:  
    処理
```

processing

「反復可能オブジェクト」から1つずつ要素を取りだして変数に代入。処理を繰り返す。
Take elements from the "Iterable Object" one by one and assign them to a variable.
The process is repeated.

```
for i in [10, 20, 30, 40, 50]:  
    print(i)
```

実行結果



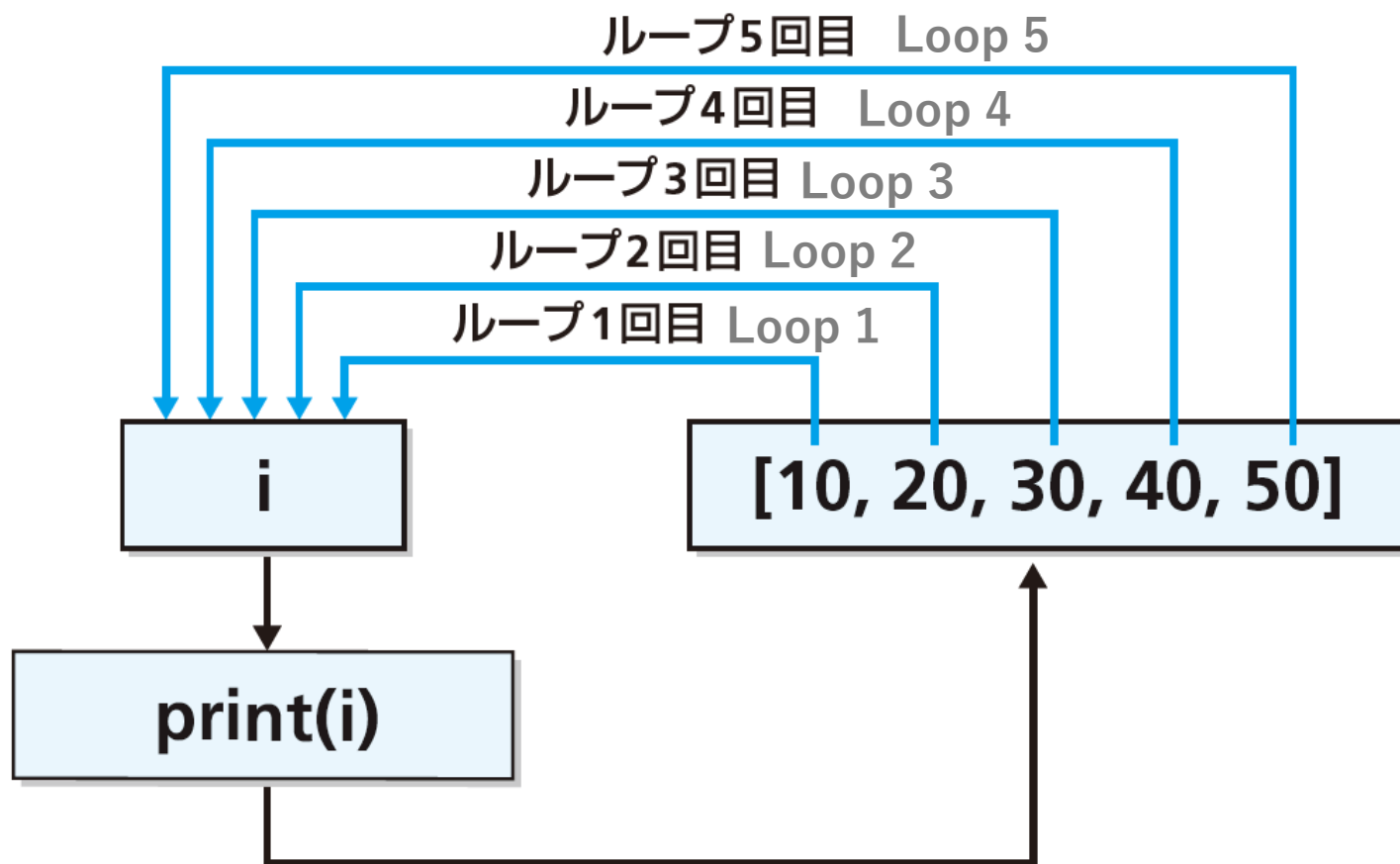
Execution Result

10
20
30
40
50

for文の処理の流れ

Process flow of for statement

```
for i in [10, 20, 30, 40, 50]:  
    print(i)
```



rangeオブジェクト range object

```
for i in range(10):  
    print(i)
```

← 変数 **i** に 0 から 9 の値が順番に代入されます

Variable i is assigned values from 0 to 9 in sequence.

実行結果 Execution Result

0
1
2
(略)
9

0 から 9 の値が 1 ずつ順番に出力されます

The values 0 to 9 are output one by one in sequence.

Range object generation method

range オブジェクトの生成方法	得られる整数の列 The resulting sequence of integers.
range(stop)	0 から「stop の値 - 1」までの整数 Integers from 0 to "stop value -1".
range(start, stop)	start の値から「stop の値 - 1」までの整数 An integer from the value of start to the value of stop -1.
range(start, stop, step)	start の値から「stop の値 - 1」までの整数。ただし、増分は step の値
	An integer from the value of start to the value of stop -1. However, the increment is the value of step.

rangeオブジェクト range object

```
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]    ← 0から9までの数字が並びます  
>>> list(range(3, 10))  
[3, 4, 5, 6, 7, 8, 9]    ← 3から9までの数字が並びます    The numbers from 3 to 9 are lined up.  
>>> list(range(1, 30, 10))  
[1, 11, 21]    ← 29を超えない範囲で1から10ずつ値が増えます  
The value is incremented from 1 to 10 in increments not exceeding 29.
```

```
for i in range(100, 201, 5):  
    print(i)    ← 100から始まり5ずつ増える値が、  
                200に達するまで順番に代入されます
```

The values are substituted in sequence,
starting at 100 and increasing by 5 until 200 is reached.

実行結果 Execution Result

```
100  
105  
110  
(略)  
200
```


ループ処理の流れの変更 Change of loop processing flow

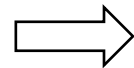
break ループの処理を中断する Interrupts the processing of the loop.

```
total = 0
for i in range(10):
    total += i
    if total > 20:
        break
print(i, total)
```

Loop1 (i=1) total = 0+1=1
Loop2 (i=2) total = 1+2=3
Loop3 (i=3) total = 3+3=6
Loop4 (i=4) total = 6+4=10
Loop5 (i=5) total = 10+5=15
Loop6 (i=6) total = 15+6=21

→20を超えたので**break** (中断)
break (interrupt) because it has exceeded 20.

実行結果



6, 21

Execution Result

ループ処理の流れの変更 Change of loop processing flow

continue ループの処理をスキップする Skip loop processing

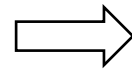
```
total = 0
for i in range(100):
    if i % 3 == 0:
        continue
    print(i)
    total += i

print('合計は', total)
```

→iが3で割り切れるときだけループ処理をスキップ

Skip the loop process only when the value of i divided by 3 is 0.

実行結果



Execution Result

```
1
2
4
5
7
8
略
...
98
合計は3267
```

ループ処理のネスト Nested loop processing

```
for a in range(1, 4):  
    print('a=', a)  
    for b in range(1, 4):  
        print('        b=', b)
```

forループの中にfor文があります

There is a for statement in the for loop.

実行結果 Execution Result

```
a= 1  
    b= 1  
    b= 2  
    b= 3  
a= 2  
    b= 1  
    b= 2  
    b= 3  
a= 3  
    b= 1  
    b= 2  
    b= 3
```

内側のループ
Inside loop

内側のループ
Inside loop

内側のループ
Inside loop

外側のループ
Outer loop

練習問題7 Question 7

課題：解答の.pyファイルを次回授業までに提出
Homework: Submit .py file of answers by next class

48000の約数はいくつあるか。for文およびif文を用いてプログラムを作成し、出力せよ。

How many 48 000 divisors are there? Create and output a program using for and if statements.

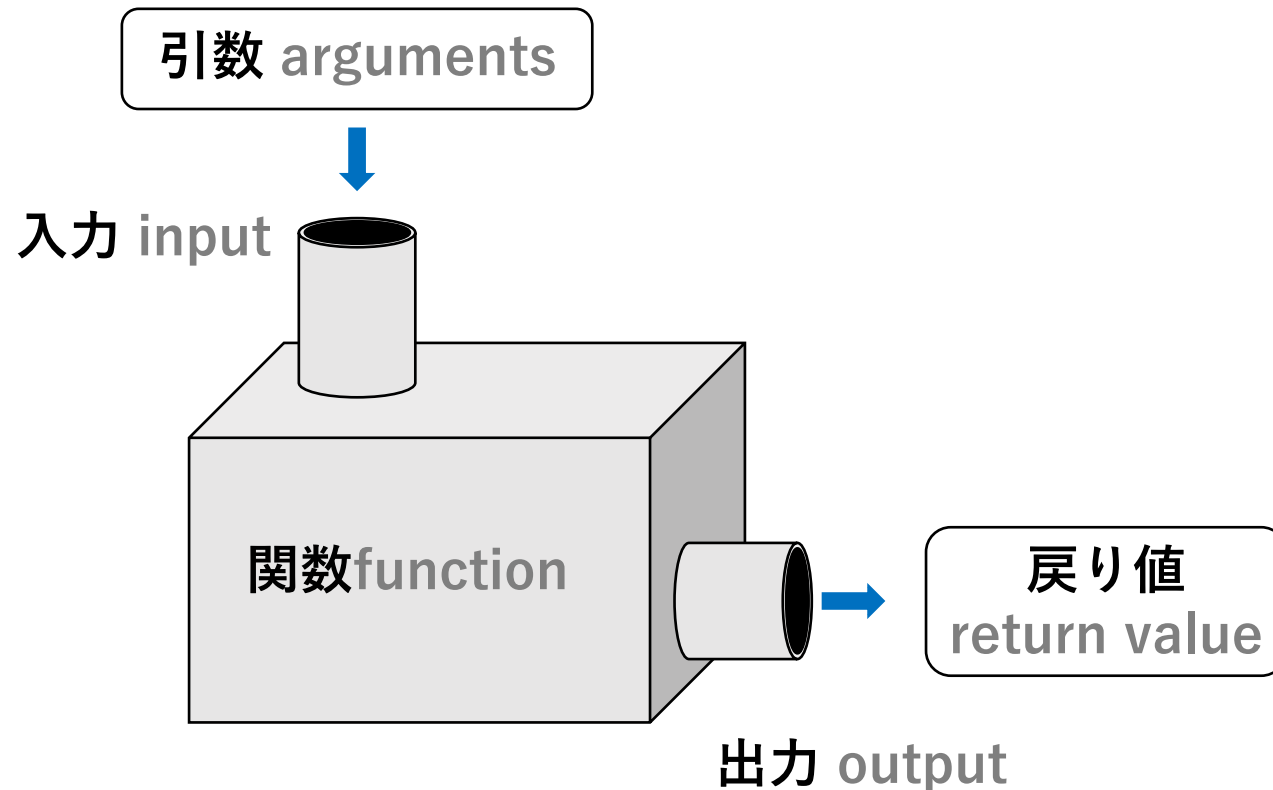
ユーザ定義関数

User-defined function



関数とは What is the function ?

- ▶ 複数の命令文を1つにまとめて名前をつけたもの
- ▶ 何度も使う命令文の集まりを管理したり、プログラムコードの見通しをよくするのに便利な仕組み
- ▶ 基本的には、引数を入力し、関数によって処理が行われ、戻り値が出力される
- ▶ A grouping of multiple imperative statements into one with a name
- ▶ Convenient mechanism for managing a collection of instruction statements that are used many times and for improving the visibility of program code
- ▶ Basically, you enter arguments, they are processed by the function, and the return value is output1



関数とは What is the function ?

2種類の関数 Two types of functions

▶ **組み込み関数**：はじめから準備されている関数

例： print関数、 id関数、 len関数、 del関数

▶ **Built-in functions**: functions prepared from scratch

Examples: print, id, len, del

▶ **ユーザー定義関数**：自分で新しく定義する関数

▶ **User-defined functions**: functions that you define yourself

構文 sentence

function name
def 関数名():
 処理
processing

ユーザー定義関数 User-defined functions

```
def say_hello():  
    print('こんにちは')  
    print('今日はよい天気ですね')
```

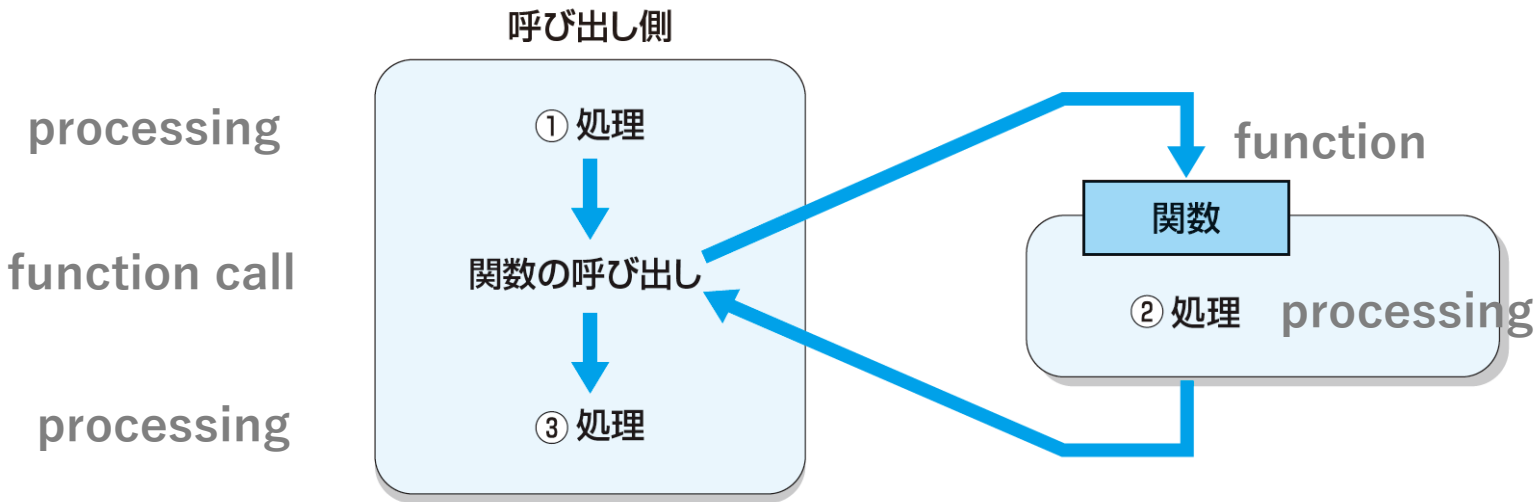
← say_hello 関数の定義 define say_hello function

```
say_hello() ← say_hello 関数の呼び出し  
Calling the say_hello function
```

実行結果 Execution Result

```
こんにちは  
今日はよい天気ですね
```


関数呼び出しの流れ Process flow of function call



```
def function_a():  
    print('function_aの処理です')
```

関数 function_a の定義です Definition of function function_a

```
def function_b():  
    print('function_bの処理です')
```

関数 function_b の定義です Definition of function function_b

```
function_a()  
function_b()
```

関数 function_a を呼び出します Call function function_a
関数 function_b を呼び出します Call function function_b

function_aの処理です
function_bの処理です

関数の定義位置 Function definition position

- 関数の定義は、関数の呼び出しの前に行われている必要がある。

The function definition must have been done prior to the function call.

```
def function_a():  
    print('function_aの処理です')
```

```
function_a()  
function_b()
```

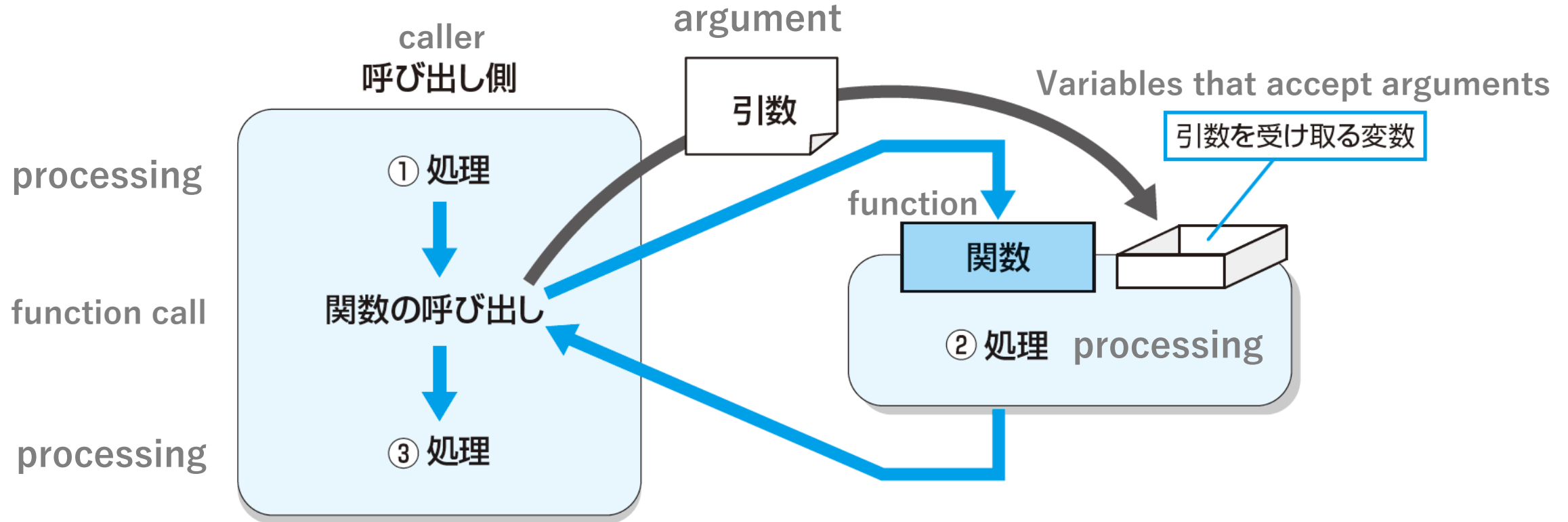
この時点では `function_b` が定義
されていないためエラーになります

```
def function_b():  
    print('function_bの処理です')
```

At this point, an error occurs
because `function_b` is not defined.

引数とは What is the argument ?

- ▶関数を呼び出すときに、関数に対して値を渡すことができる。
- ▶渡される値を**引数**という。
- ▶When calling a function, a value can be passed to the function.
- ▶The value passed is called the argument.



引数のある関数 Functions with arguments

```
def countdown(start):
```

関数の後ろのカッコ()の中の変数で値を受け取る
Receives values in variables in parentheses () after the function

```
    print('関数が受け取った値:', start)
```

Value received by the function

```
    print('カウントダウンをします')
```

countdown

```
    counter = start
```

← 受け取った値を使用する

```
    while counter >= 0:
```

Use the received value

```
        print(counter)
```

```
        counter -= 1
```

```
countdown(3) ← 値3を引数にして関数を呼び出す
```

```
countdown(10) Call a function with the value 3 as an argument
```

実行結果 Execution Result

```
関数が受け取った値: 3
```

```
カウントダウンをします
```

```
3
```

```
2
```

```
1
```

```
0
```

```
関数が受け取った値: 10
```

```
カウントダウンをします
```

```
10
```

```
9
```

```
(中略)
```

```
1
```

```
0
```

引数が2つある関数 Functions with 2 arguments

引数をカンマ区切りで並べる (引数列)

Comma-separated list of arguments (argument columns)

```
def countdown(start, end):
```

```
    print('1つ目の引数で受け取った値:', start)
```

```
    print('2つ目の引数で受け取った値:', end)
```

```
    print('カウントダウンをします')
```

```
    counter = start
```

```
    while counter >= end:
```

```
        print(counter)
```

```
        counter -= 1
```

```
countdown(7, 3) ← 2つの値を引数にして関数を呼び出す
```

Call a function with two values as arguments

変数名を指定した呼び出し
(キーワード引数)

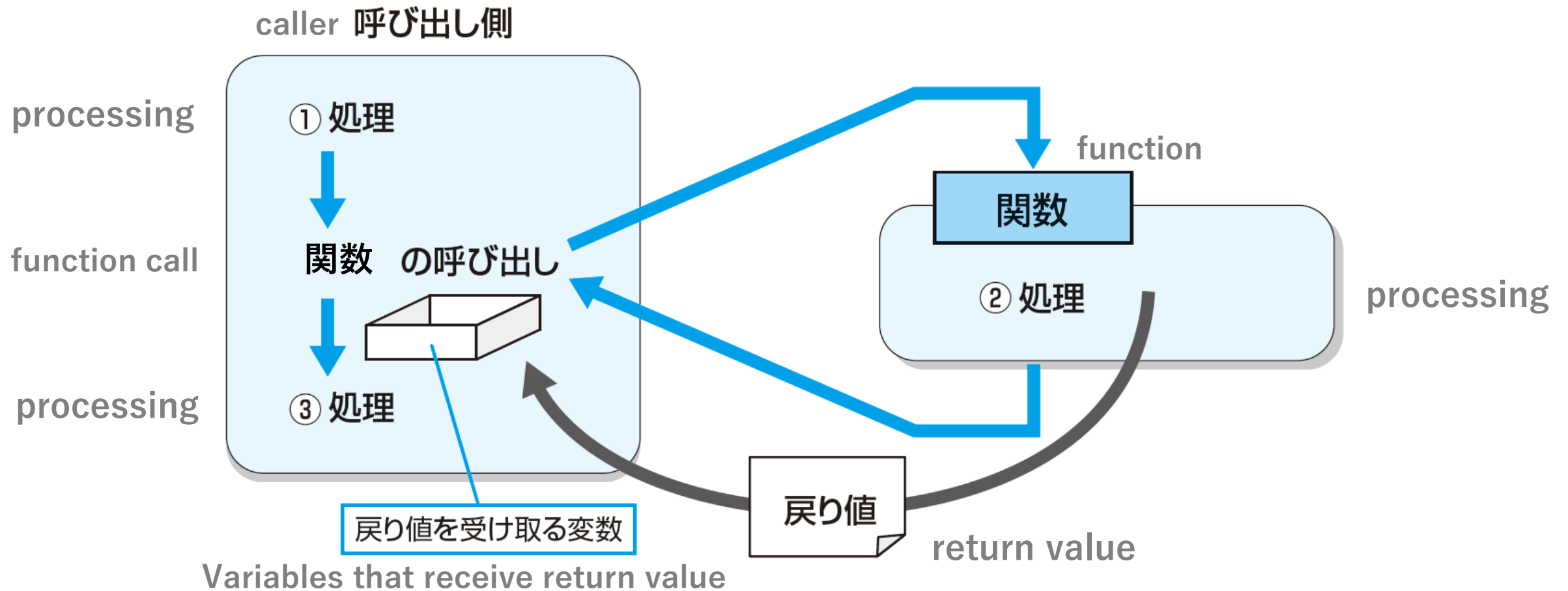
Calls with variable names
(keyword argument)

```
countdown(start=7, end=3)
```

```
countdown(end=3, start=7)
```

戻り値とは What is the return value ?

- ▶関数で処理した結果の値を、呼び出し元に戻すことができる。
- ▶戻される値のことを「戻り値」(返り値)という。
- ▶The value of the result of processing in a function can be returned to the caller.
- ▶The value returned is called the "return value" (return value).



戻り値のある関数 function with return value

・値を戻すには、関数の中に To return a value, write

```
return 値
```

と記述する。 in the function.

```
def circle_area(r):  
    return r * r * 3.14  
  
a = circle_area(2.5)  
print('半径2.5の円の面積は', a)
```

The area of a circle of radius 2.5 is

グラフの保存 save a graph

MATLABスタイルインターフェース：

MATLAB-style interface

```
plt.savefig("file_name")
```

オブジェクト指向インターフェース：

Object-oriented interface

```
figure.savefig("file_name")
```

保存できる形式 Formats that can be saved

emf, eps, jpeg, pdf, png, ps, raw, rgba, svg, svgz, tif, tiff

解像度を指定して保存 Save with specified resolution

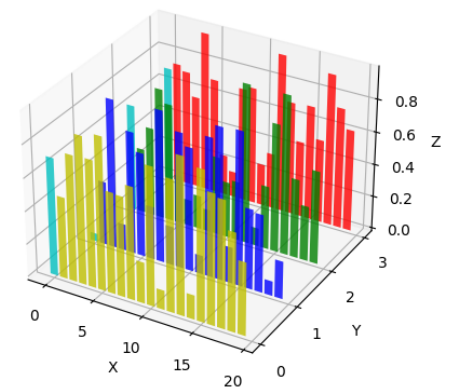
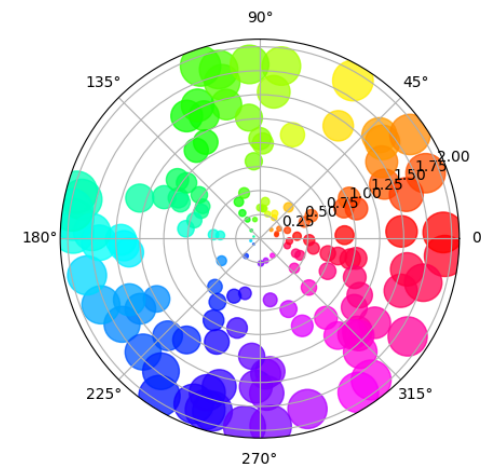
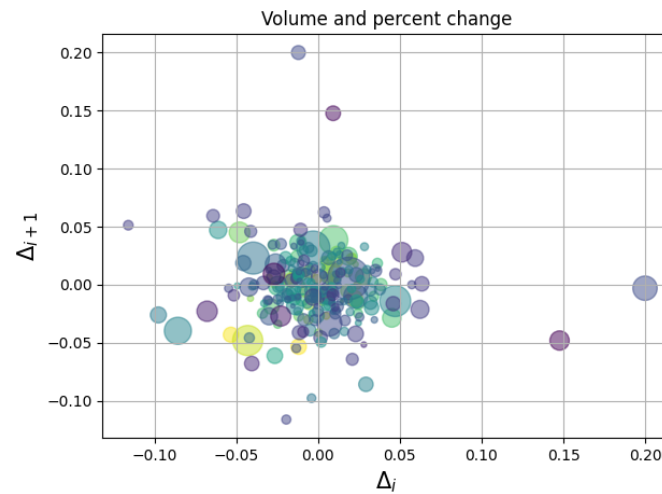
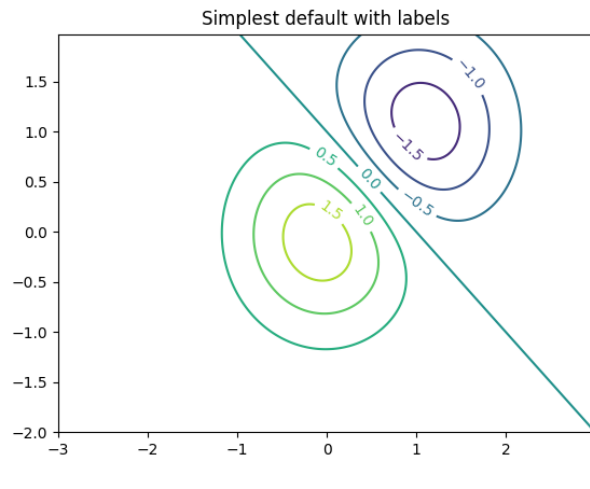
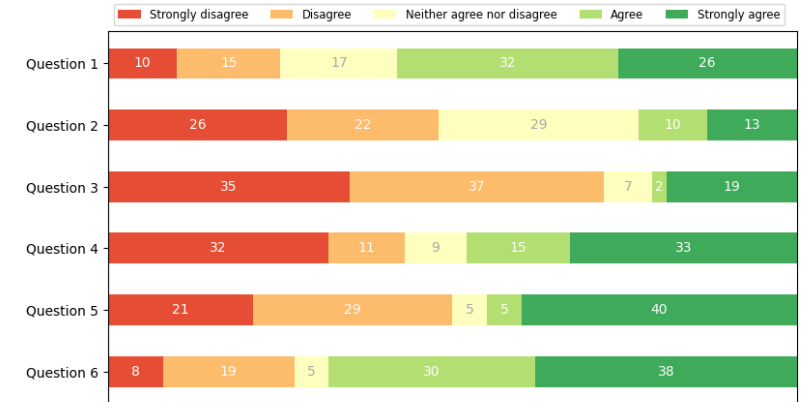
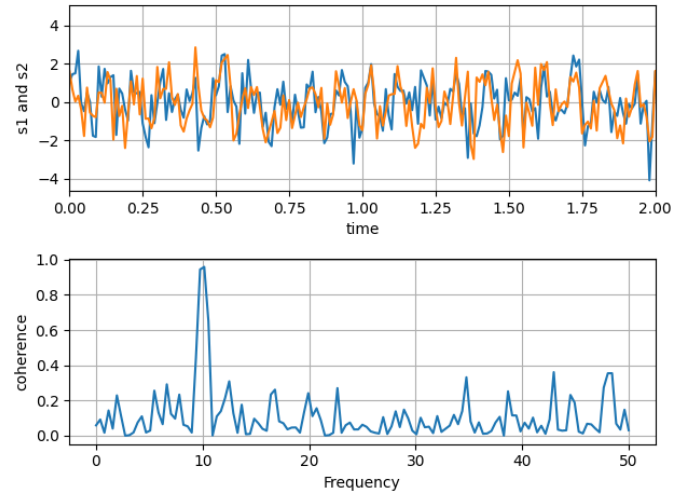
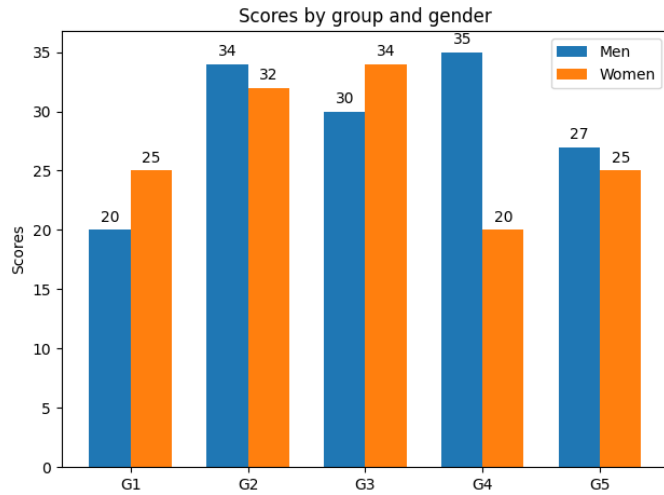
```
fig.savefig("300_dpi_scatter.png", dpi=300)
```


Matplotlibによるグラフの描画

Drawing Graphs with Matplotlib

Matplotlib：グラフを描くライブラリ

Libraries for drawing graphs



Matplotlibのインポートとプロット Importing and plotting Matplotlib

Matplotlibの中のモジュールの一つであるmatplotlib.pyplotをインポートする。
(基本的にはこのモジュールのみでOK)

Import matplotlib.pyplot, one of the modules in Matplotlib.(Basically, you only need this module)

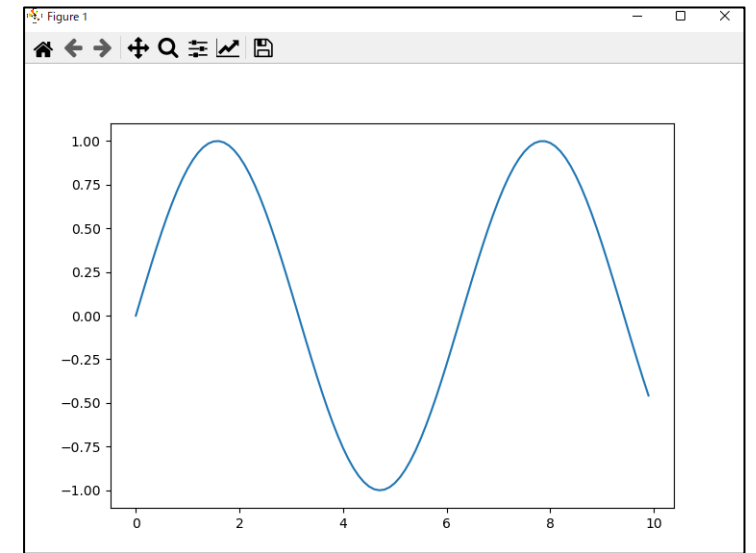
```
import matplotlib.pyplot as plt  
import numpy as np
```

```
x = np.arange(0,10,0.1)  
y = np.sin(x)
```

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

- plt.plot()関数で折れ線グラフ描画
- plt.show()関数で表示

Plotting a line chart with plt.plot() function
Display with plt.show() function



Matplotlibの文法 Matplotlib Syntax

Matplotlibでグラフを描画するには2通りの方法がある。同じものが描画される。

There are two ways to draw a graph in Matplotlib. The same thing is drawn.

MATLABスタイルインターフェース

簡単なプロットを描画する際に高速かつ便利

MATLAB-style interface

Fast and convenient for drawing simple plots

オブジェクト指向インターフェース

図を細かく制御したい場合に便利

公式はこちらを推奨している

Object-oriented interface

Useful for fine control of diagrams

Official recommendation

MATLABスタイルインターフェース

MATLAB-style interface

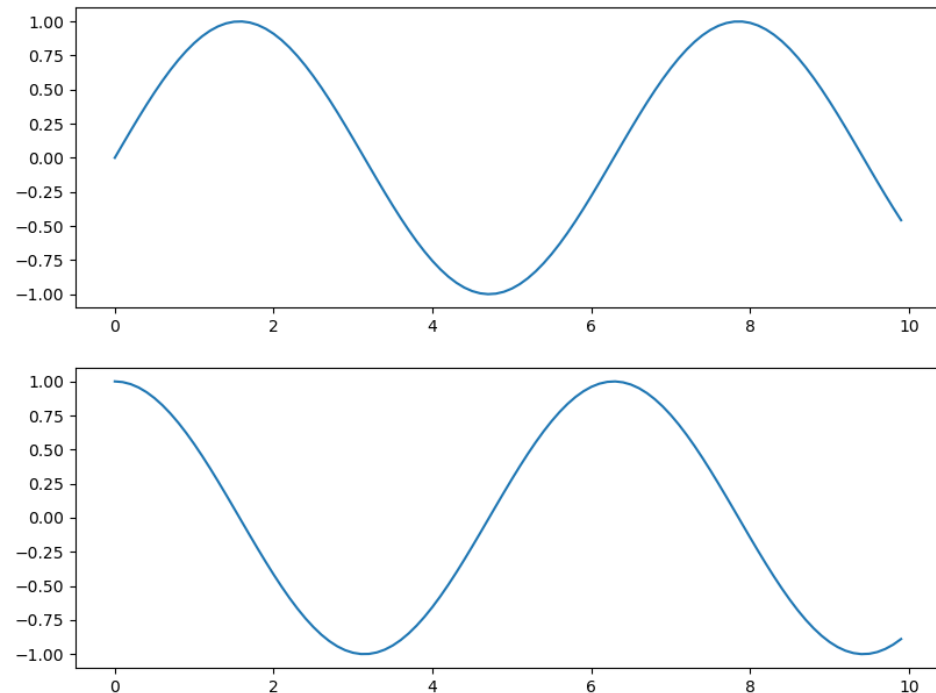
```
import matplotlib.pyplot as plt  
import numpy as np
```

```
x = np.arange(0,10,0.1)
```

```
plt.figure(1)
```

```
plt.subplot(211)  
plt.plot(x,np.sin(x))
```

```
plt.subplot(212)  
plt.plot(x,np.cos(x))  
plt.show()
```



オブジェクト指向インターフェース

Object-oriented interface

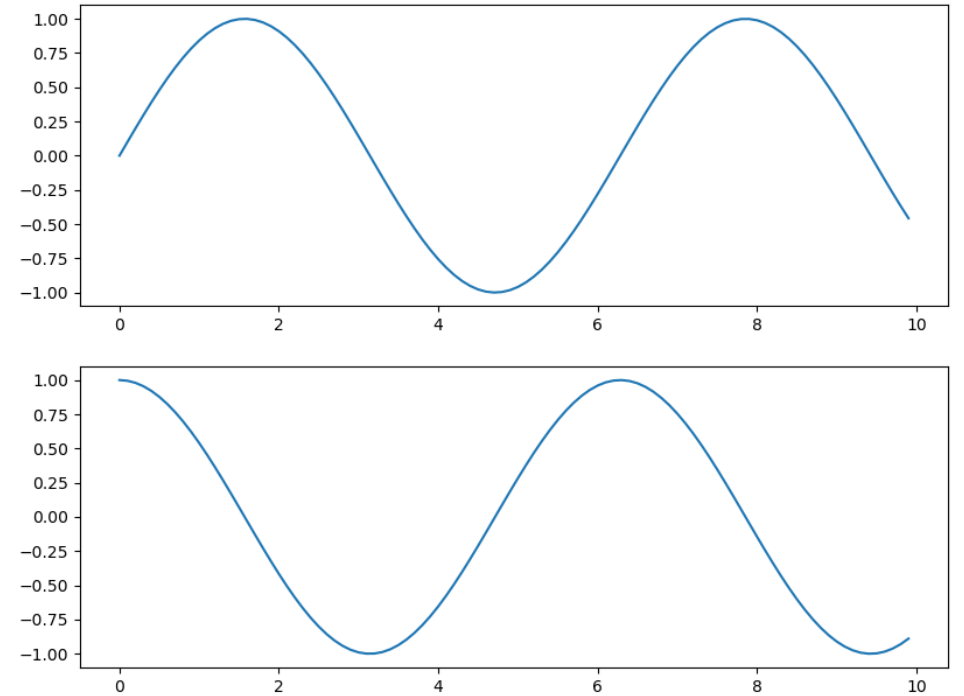
```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.arange(0,10,0.1)
```

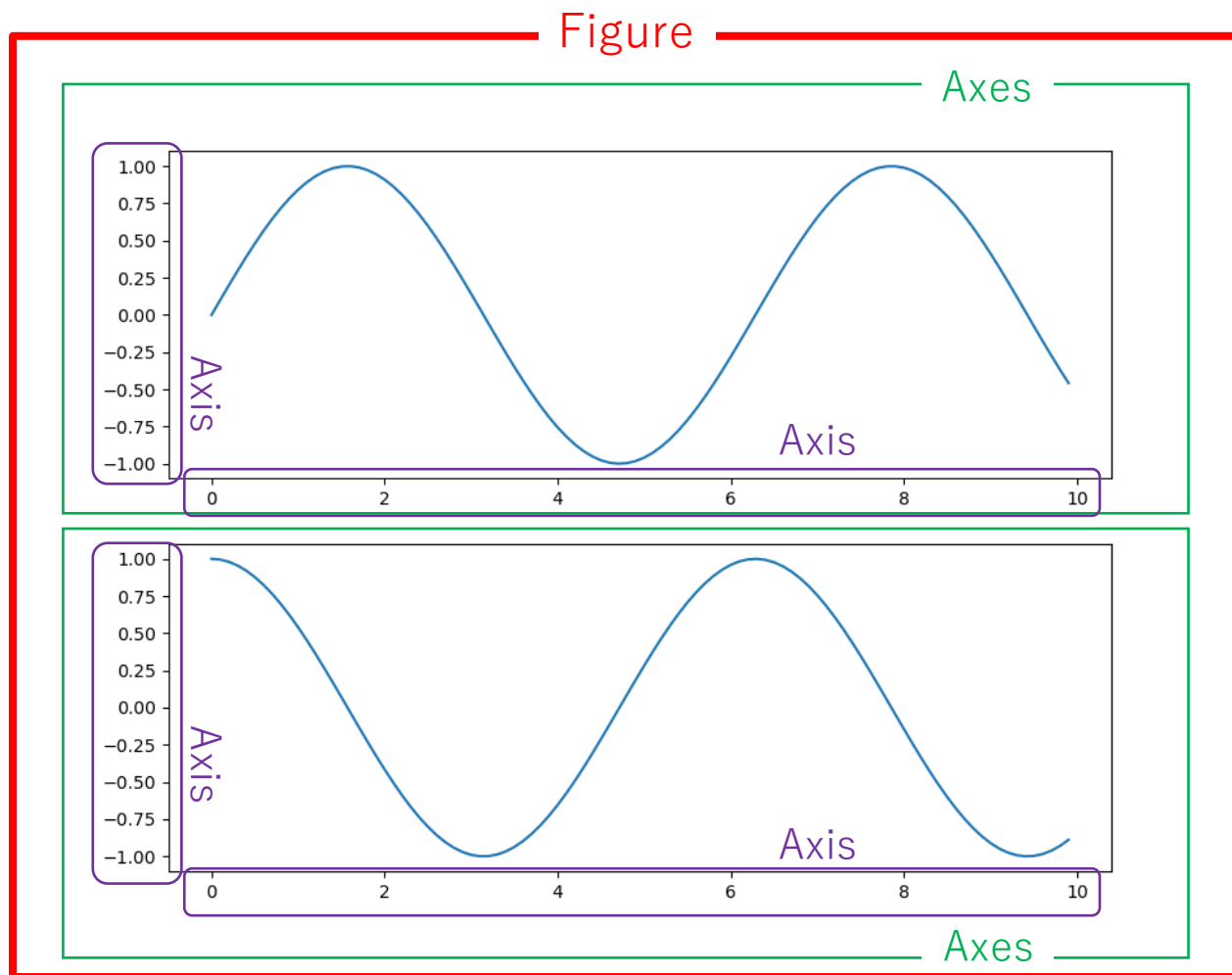
```
fig = plt.figure()
```

```
ax1 = fig.add_subplot(2,1,1)
ax1.plot(x,np.sin(x))
```

```
ax2 = fig.add_subplot(2,1,2)
ax2.plot(x,np.cos(x))
plt.show()
```



Matplotlibの階層構造 Matplotlib Hierarchy

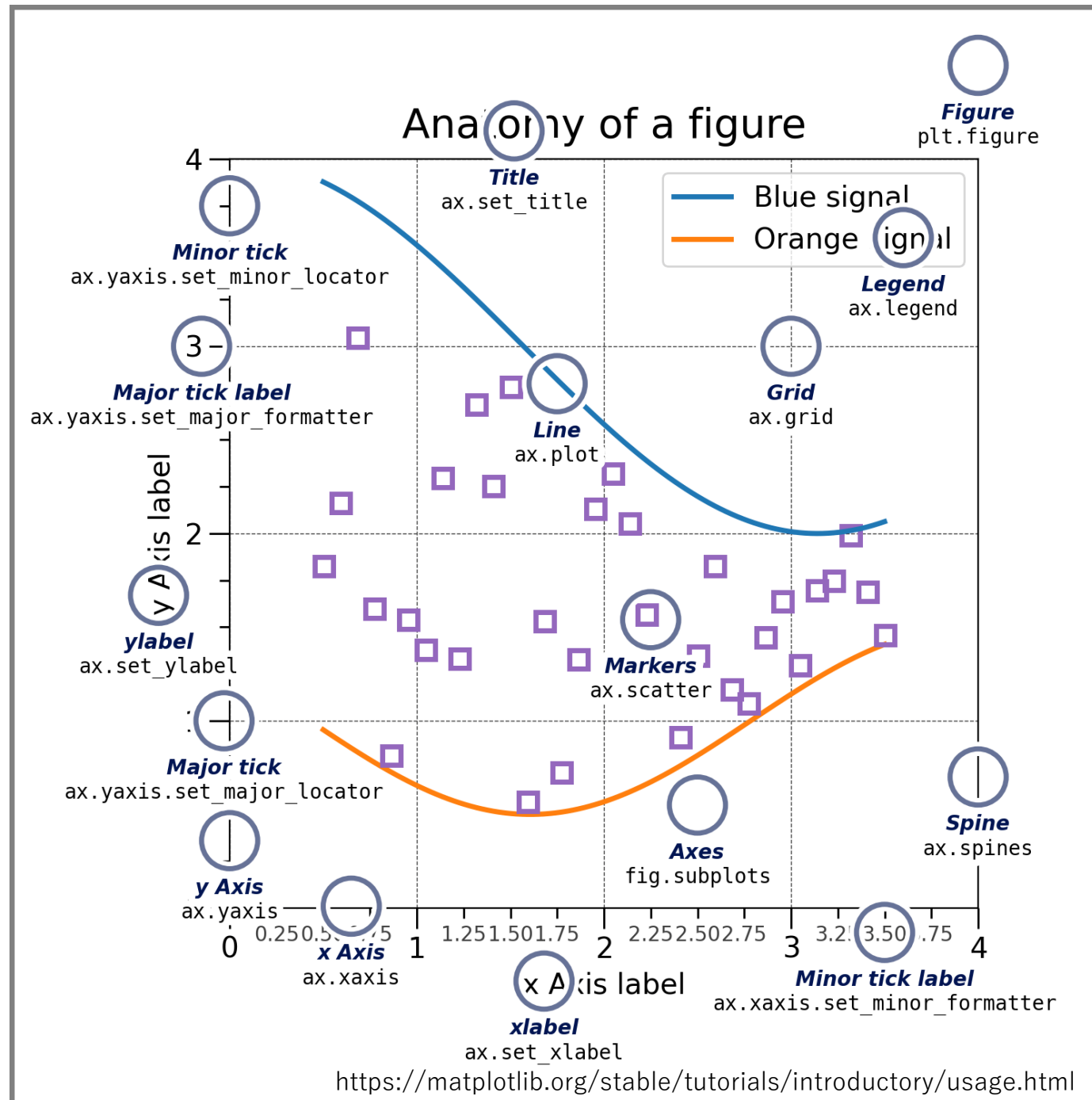


- FigureオブジェクトにAxesオブジェクトが属している
- AxesオブジェクトにはAxisオブジェクトが属している

Axes object belongs to Figure object
Axis object belongs to Axes object

Figureの構成

Composition of Figure



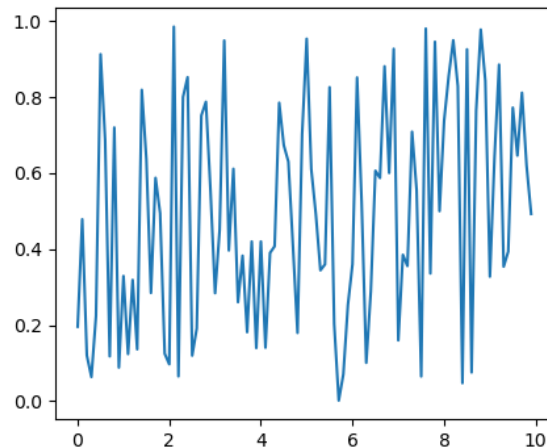
グラフの種類 Graph Type

```
import matplotlib.pyplot as plt
import numpy as np
```

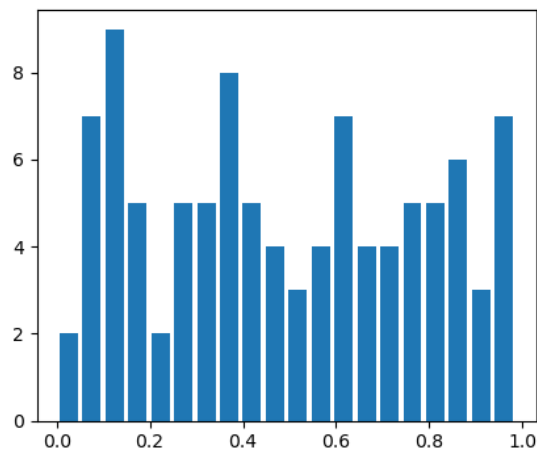
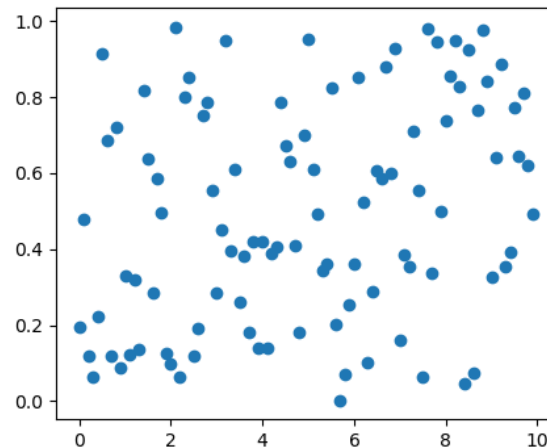
```
x = np.arange(0,10,0.1)
y = np.random.rand(len(x))
```

```
fig = plt.figure()
ax1 = fig.add_subplot(2,2,1)
ax1.plot(x,y)
ax2 = fig.add_subplot(2,2,2)
ax2.scatter(x,y)
ax3 = fig.add_subplot(2,2,3)
ax3.hist(y,rwidth=0.8, bins=20)
ax4 = fig.add_subplot(2,2,4)
ax4.boxplot(y)
plt.show()
```

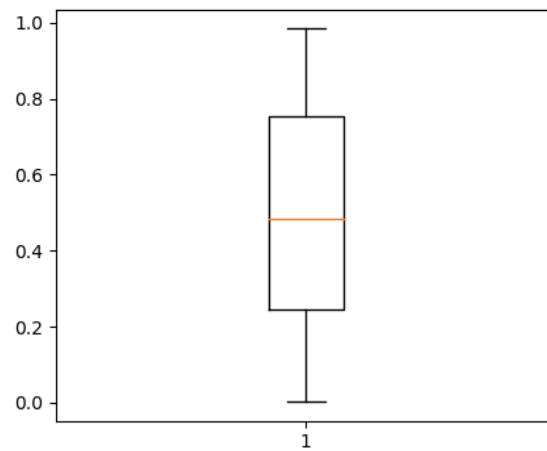
plot (折れ線)



scatter (散布図)



hist (ヒストグラム)



box (箱ひげ図)

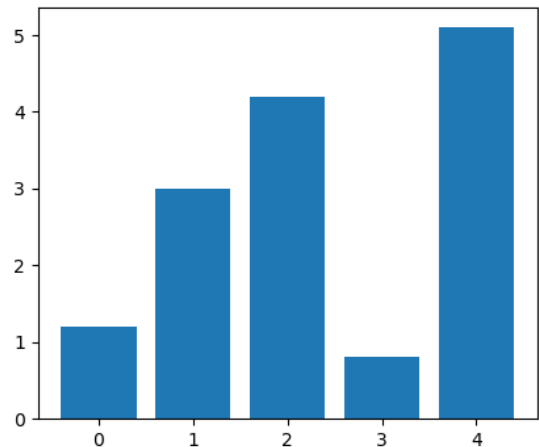
グラフの種類 Graph Type

```
import matplotlib.pyplot as plt
import numpy as np
```

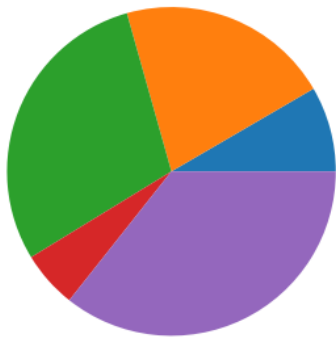
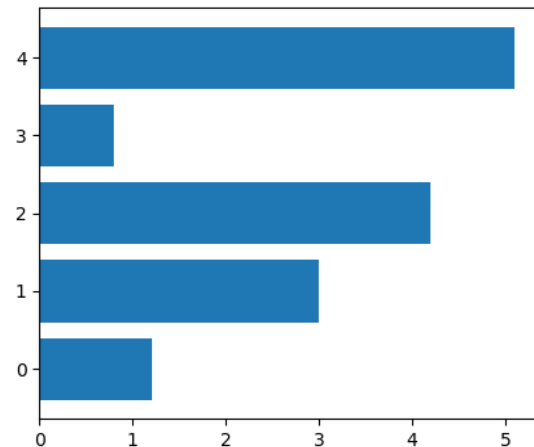
```
x = list(range(5))
y = [1.2, 3, 4.2, 0.8, 5.1]
```

```
fig = plt.figure()
ax1 = fig.add_subplot(2,2,1)
ax1.bar(x,y)
ax2 = fig.add_subplot(2,2,2)
ax2.barh(x,y)
ax3 = fig.add_subplot(2,2,3)
ax3.pie(y)
plt.show()
```

bar (棒グラフ)



barh (棒グラフ横向き)



pie (円グラフ)

線の色 Line color

```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.arange(0,10,0.1)
fig = plt.figure()
```

```
ax = fig.add_subplot(1,1,1)
ax.plot(x, np.sin(x-0), color="blue")
ax.plot(x, np.sin(x-1), color="g")
ax.plot(x, np.sin(x-2), color="0.75")
ax.plot(x, np.sin(x-3), color="#FFDD44")
ax.plot(x, np.sin(x-4), color=(1.0,0.2,0.3))
ax.plot(x, np.sin(x-5), color="chartreuse")
plt.show()
```

色の名前で指定

カラーコード(rgbcmk)による指定

0から1の間のグレースケールを指定

16進コードで指定(RRGGBBを00からFFで表す)

0から1のRGB値をタプルで指定

サポートされているHTMLカラーネームで指定

Designation by color name

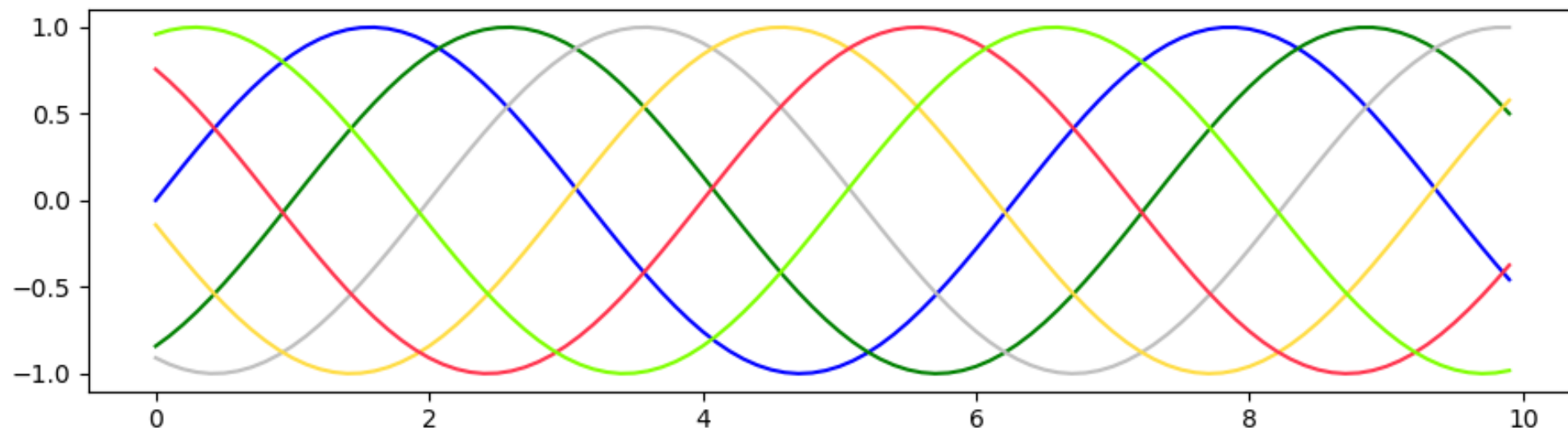
Designation by color code (rgbcmk)

Specify grayscale between 0 and 1

Designation by hexadecimal code

Specify RGB values between 0 and 1 as a tuple

Specify by supported HTML color names



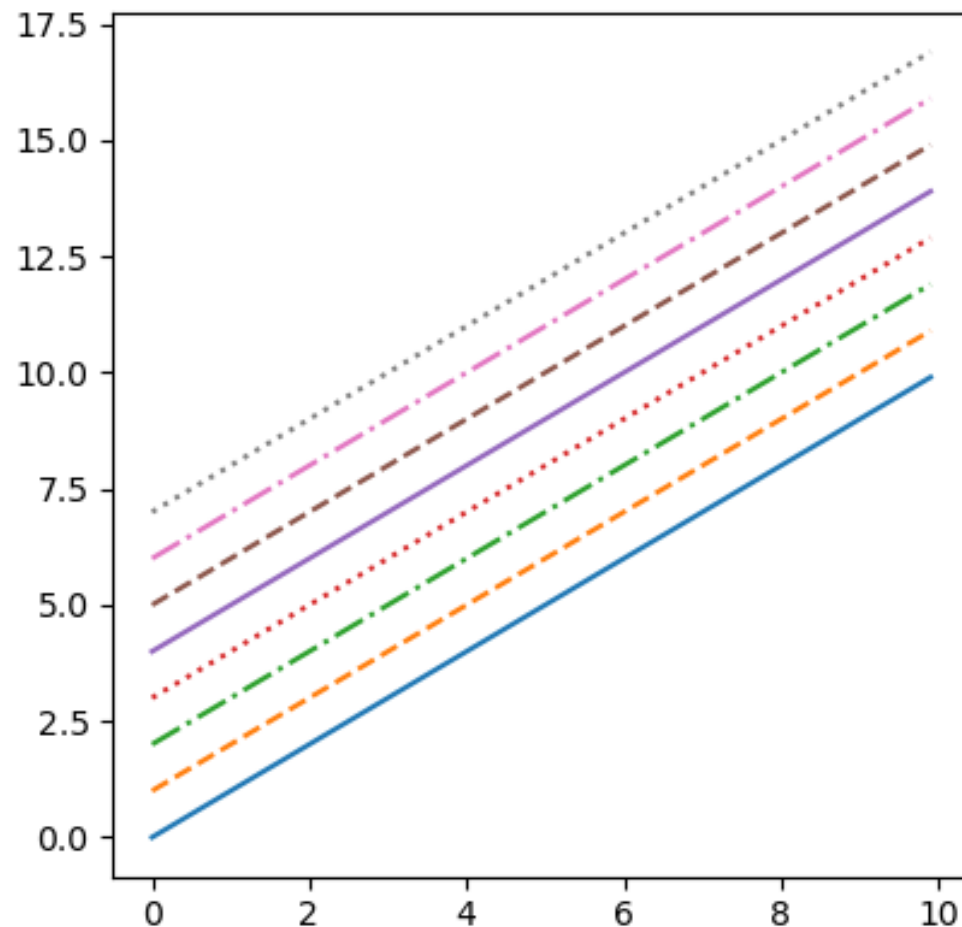
線のスタイル Line style

```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.arange(0,10,0.1)
fig = plt.figure()
```

```
ax = fig.add_subplot(1,1,1)
ax.plot(x, x+0, linestyle="solid")
ax.plot(x, x+1, linestyle="dashed")
ax.plot(x, x+2, linestyle="dashdot")
ax.plot(x, x+3, linestyle="dotted")
ax.plot(x, x+4, linestyle="-")
ax.plot(x, x+5, linestyle="--")
ax.plot(x, x+6, linestyle="-.")
ax.plot(x, x+7, linestyle=":")
plt.show()
```

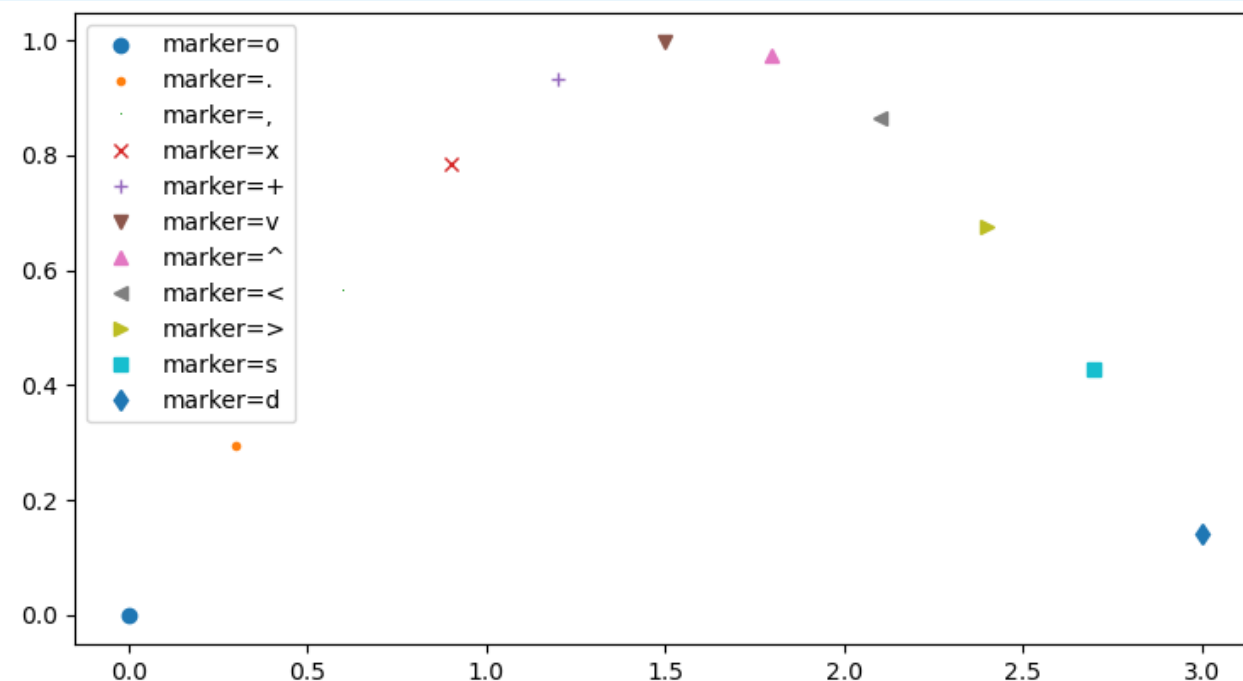
実線
破線
一点鎖線
点線
実線
破線
一点鎖線
点線



プロットのシンボル Plot symbols

```
import matplotlib.pyplot as plt
import numpy as np
```

```
marker_list = ["o", ".", ",", "x", "+", "v", "^", "<", ">", "s", "d"]
x = np.linspace(0,3,len(marker_list))
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
for i, k in enumerate(x):
    ax.plot(k,np.sin(k),marker_list[i],label=f"marker={marker_list[i]}")
plt.legend()
plt.show()
```



Axesオブジェクトのメソッド Methods of the Axes object

Setting the Title

x-axis label setting

y-axis label setting

x-axis range setting

y-axis range setting

Setting of x-axis tick marks

Setting of y-axis tick marks

Display scale lines

Display legend

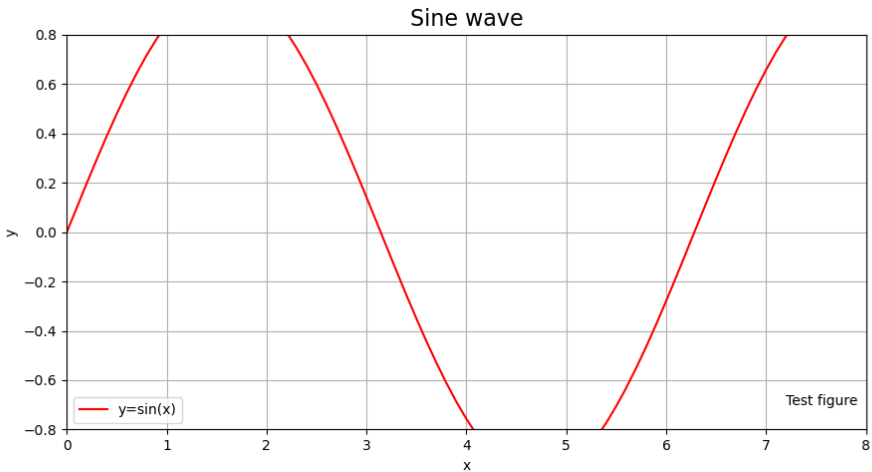
Display text

やりたいこと	メソッド
タイトルを設定	axes.set_title ()
x軸ラベルの設定	axes.set_xlabel ()
y軸ラベルの設定	axes.set_ylabel ()
x軸範囲の設定	axes.set_xlim ()
y軸範囲の設定	axes.set_ylim ()
x軸の目盛り設定	axes.set_xticks()
y軸の目盛り設定	axes.set_yticks()
目盛り線を表示	axes.grid()
凡例を表示	axes.legend()
テキストの表示	axes.text()

それぞれのメソッドには引数があって、文字の大きさや色などを指定できる。

Each method has arguments to specify the size and color of the text.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0,10,0.1)
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.plot(x, np.sin(x), label="y=sin(x)", color="red")
ax.set_title("Sine wave", fontsize=16)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_xlim(0,8)
ax.set_ylim(-0.8,0.8)
ax.set_xticks(list(range(0,9)))
ax.set_yticks([-0.8,-0.6,-0.4,-0.2,0.0,0.2,0.4,0.6,0.8])
ax.grid()
ax.legend()
ax.text(7.2, -0.7, "Test figure", size=10)
plt.show()
```



グラフの保存 save a graph

MATLABスタイルインターフェース：

MATLAB-style interface

```
plt.savefig("file_name")
```

オブジェクト指向インターフェース：

Object-oriented interface

```
figure.savefig("file_name")
```

保存できる形式 Formats that can be saved

emf, eps, jpeg, pdf, png, ps, raw, rgba, svg, svgz, tif, tiff

解像度を指定して保存 Save with specified resolution

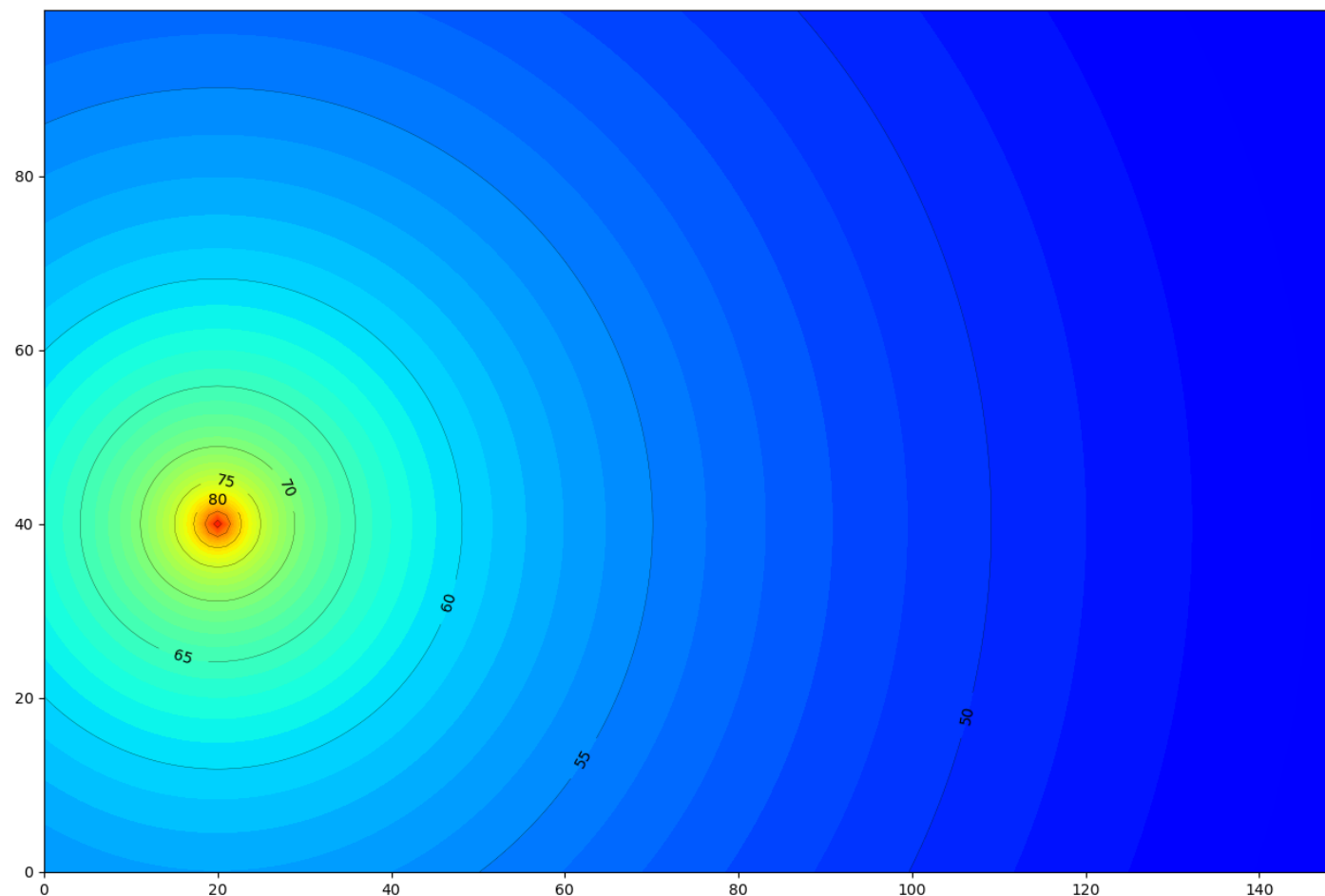
```
fig.savefig("300_dpi_scatter.png", dpi=300)
```

発展：contour関数による等高線の描画

Advanced: Drawing Contour Lines with the contour function

例：点音源からの距離減衰

Example: Distance attenuation from a point source



```
import numpy as np
import matplotlib.pyplot as plt
```

```
# 計算エリアの設定(m)
```

```
length_x = 150.0
```

```
length_y = 100.0
```

```
# 音源点(sp)の設定
```

```
sp_x = 20.0
```

```
sp_y = 40.0
```

```
sp_z = 0.5
```

```
Lw = 100.0 #dB
```

```
# 受信点(rp)の設定
```

```
calc_interval = 1.0 #受信点を設置する間隔(m)
```

```
rp_x = np.arange(0,length_x,calc_interval)
```

```
rp_y = np.arange(0,length_y,calc_interval)
```

```
rp_x,rp_y = np.meshgrid(rp_x,rp_y) #メッシュデータの生成
```

```
rp_z = 1.2
```

```
# 受信点でのレベル算出
```

```
d = np.sqrt((sp_x-rp_x)**2 + (sp_y-rp_y)**2 + (sp_z-  
rp_z)**2)
```

```
L = Lw - 11 - 20 * np.log10(d)
```

```
fig = plt.figure()
```

```
ax = fig.add_subplot(111)
```

```
ax = plt.contour(rp_x,rp_y,L,np.arange(40,100,5.0),  
colors="black",linewidths=0.2)
```

```
ax.clabel(fmt='%1.0f')
```

```
ax =
```

```
plt.contourf(rp_x,rp_y,L,np.arange(40,100,1.0),cmap="jet")
```

```
plt.colorbar()
```

```
plt.show()
```