




# Python入門 第2回

総合科学研究会

2024.4.18



# 今回の内容

- 関数やモジュールについて
- 文字列の処理

# 関数

Pythonでは算術演算のみならず、様々な便利ツールを使うことが出来る。  
その1つが関数である。

例: `abs(x)`:絶対値を返す関数

🔍	✓ 0秒	[1] <code>abs(-100)</code>
{x}		100

`print(x1, x2,...)`: `x1, x2,...` を順に表示する関数

	✓ 0秒	[4] <code>print(10, "moji")</code>
<>		10 moji

# 関数の使い方

関数を呼び出す際には、引数が必要になる。

`abs(x)`なら`x`が引数であり、`print(v0, v1, v2, v3)`なら`v0, v1, v2, v3`が引数となる。引数には変数を指定する事も可能。

✓  
0秒

```
[3] num = 5  
    abs(num * -5)
```

25

関数は実行後、基本的に値を1つ返す。これを戻り値もしくは返り値という。

`abs(x)`の返り値は $|x|$ である。

## 関数の使い方・補足

関数を他の算術演算と組み合わせて使う事も出来る。

A screenshot of a Python REPL (Read-Eval-Print Loop) interface. On the left, there is a vertical sidebar with a key icon at the top and a folder icon below it. The main area shows a prompt '[2]' followed by the expression 'abs(-20) \* 3 / 2'. The result '30.0' is displayed below the expression. A green checkmark and '0秒' (0 seconds) are visible next to the prompt, indicating a successful and fast execution.

```
[2] abs(-20) * 3 / 2
30.0
```

- なおプログラミング言語における関数と数学における関数という用語には若干の違いがある。

例えばベクトル関数 $f(v_1, v_2) = (v_1 + v_2)i + (v_1 - v_2)j$ は2つの引数に対して2つの値を返すが、プログラミング言語における関数では1つの値しか返すことが出来ない。

# モジュールについて

Pythonにおいてはモジュールというものを入れる事で色々な関数を使うことが出来る。

例えばmathモジュールを導入すると三角関数や対数関数等を使うことが出来たり、matplotlibモジュールを導入すると簡単にグラフを描画出来たりする。

モジュールを導入するときにはコードの先頭に**import** (モジュール名)と書いておく必要がある。また、モジュールの関数を呼び出す際はモジュール名.関数名という記法を使う。

✓  
0 秒

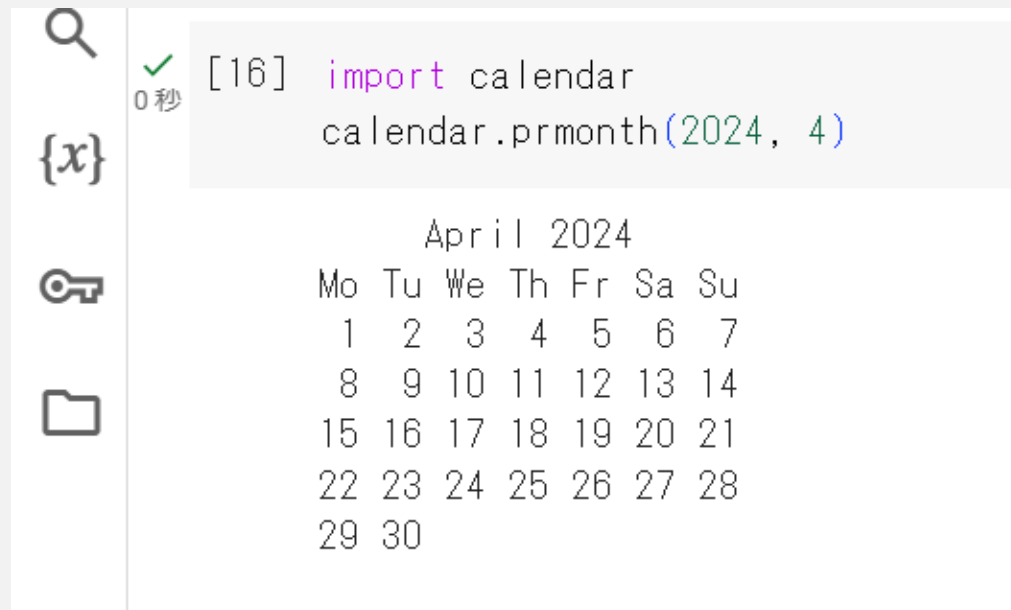
```
[5] import math  
     math.sqrt(2.0)
```

```
1.4142135623730951
```

# モジュールの例

- Calendar

カレンダーを表示する関数`prmonth(year, month)`を使うことが出来る。



The screenshot shows a Jupyter Notebook interface. On the left is a sidebar with icons for search, variables, keys, and files. The main area displays a code cell with the following content:

```
[16]: import calendar
      calendar.prmonth(2024, 4)
```

Below the code, the output of the `prmonth` function is shown, displaying the calendar for April 2024:

```
April 2024
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

## 参考

- mathをimportすることで使える関数など

math.exp(x): $e^x$

math.log(x): $\ln(x)$  自然対数

math.log(x,a): $\log_a x$

math.pow(x,y): $x^y$

math.sin(x): $\sin(x)$   $x$ はラジアンであることに注意

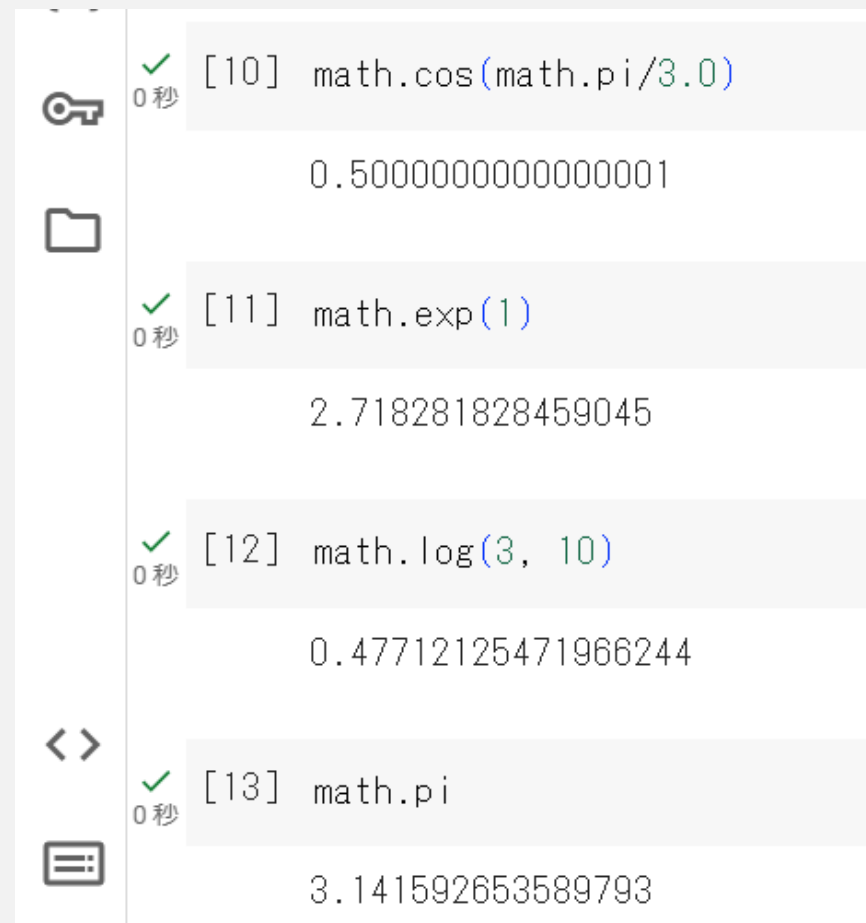
math.cos(x): $\cos(x)$

math.tan(x): $\tan(x)$

math.radians(x):度数表記→ラジアン表記に変換

math.pi:  $\pi$ (定数)

math.e:  $e$ (自然対数の底)



A screenshot of a Python REPL (Jupyter Notebook) showing four code cells. Each cell has a green checkmark icon and a '0秒' (0 seconds) execution time. The first cell shows `math.cos(math.pi/3.0)` returning `0.5000000000000001`. The second cell shows `math.exp(1)` returning `2.718281828459045`. The third cell shows `math.log(3, 10)` returning `0.47712125471966244`. The fourth cell shows `math.pi` returning `3.141592653589793`. The left sidebar shows icons for a key, a folder, and a list.

```
[10] math.cos(math.pi/3.0)
0.5000000000000001

[11] math.exp(1)
2.718281828459045

[12] math.log(3, 10)
0.47712125471966244

[13] math.pi
3.141592653589793
```



## 演習問題②

- 次の計算を実行してみてください。

(1)  $\log_2 3$

(2)  $\tan(\frac{\pi}{6})$

(3)  $11^{11}$

(4)  $\cos(1^\circ)$

(5)  $\frac{e+e^{-1}}{2}$  (math.cosh(1)の値との比較も)

# 文字列処理

# 文字列について

これまでは基本的に数値データを取り上げてきたが、プログラミング言語では文字データも扱うことが多い。

Pythonでは、ダブルクォーテーション" "もしくはシングルクォーテーション' 'で文字を囲う事で文字データ(文字型)として処理される。

{x}	✓ 0秒	[17] "Hello world!"
🔑		'Hello world!'

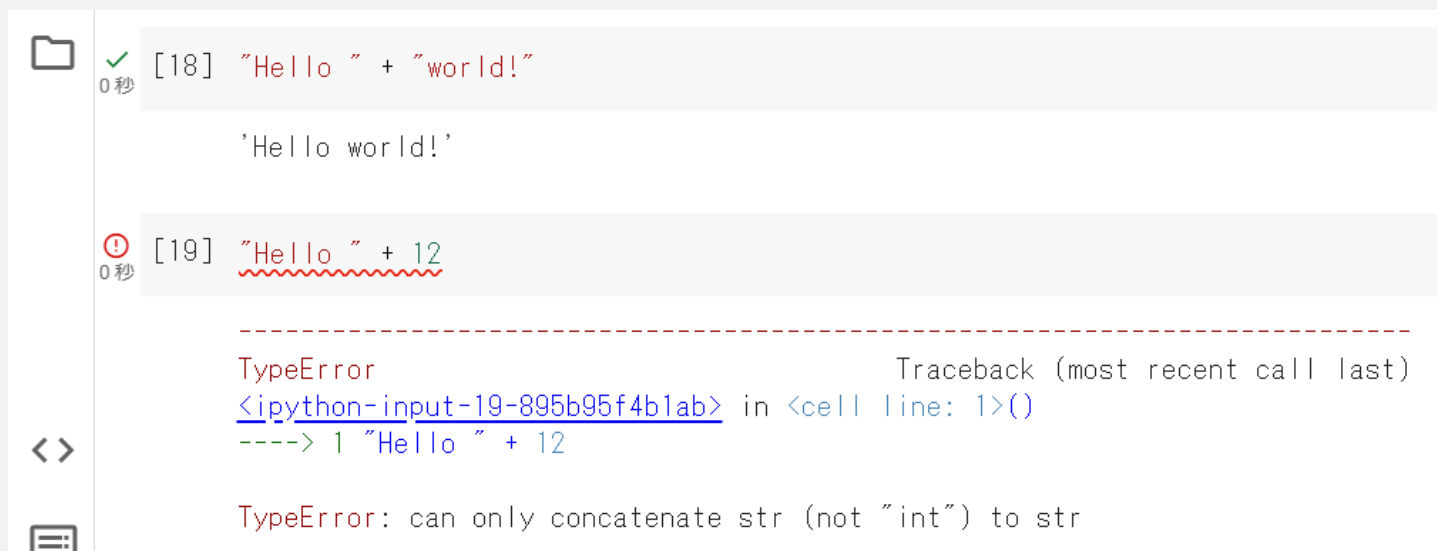
# 文字列に対する演算

文字列に対して足し算+を行うとどうなるか？

> 文字列が結合される。

例えば"Hello " + "world!" → "Hello world!"となる。

ただし、数値データと文字列との足し算はエラーになるので注意。



```
[18] "Hello " + "world!"  
  
'Hello world!'  
  
[19] "Hello " + 12  
  
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-19-895b95f4b1ab> in <cell line: 1>()  
----> 1 "Hello " + 12  
  
TypeError: can only concatenate str (not "int") to str
```

## データ型の違い

整数型(int)と浮動小数点型(float)同士の四則演算は定義されていたが、文字型と数値データとの演算はエラーとなる。

数値の12と数値の34を掛ける( $12 * 34$ )と408となるが、クォーテーションで囲っただけの"12" \* "34"は文字列同士の掛け算とみなされてエラーとなる。

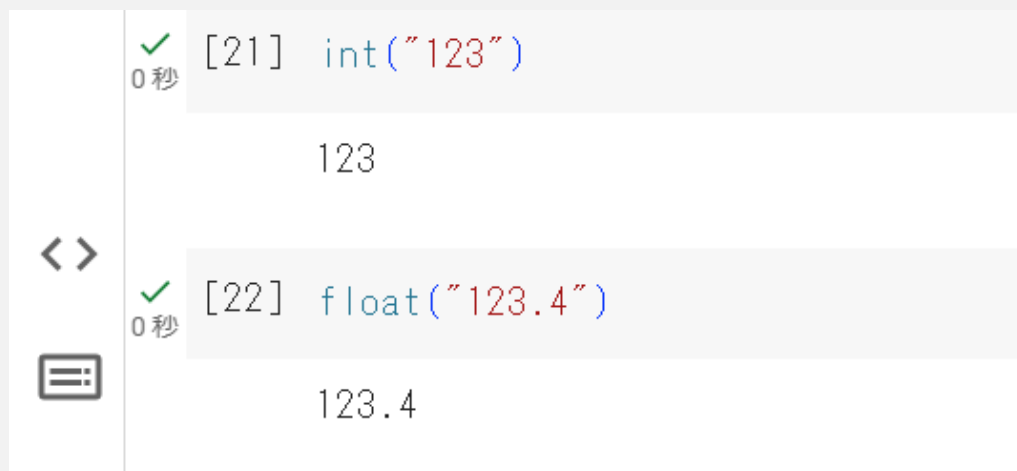
このようにデータ型が異なるデータというのは基本的に別物のデータであり、それぞれのデータ型専用の演算や関数を使用しなければならない。

# 文字列と数値の相互変換

データ型は別とはいえ、数値の123と文字列の"123"でそれぞれ別の操作をしなければならないというのは面倒

→ **文字列が数字からなる場合に限り**、数値に変換する(あるいは逆の変換)関数int()やfloat()等が用意されている。

数字からなる文字列sに対して int(s)とするとsの中身の数字が整数型として返される。  
またfloat(s)を使うと浮動小数点型(float)として返してくれる。



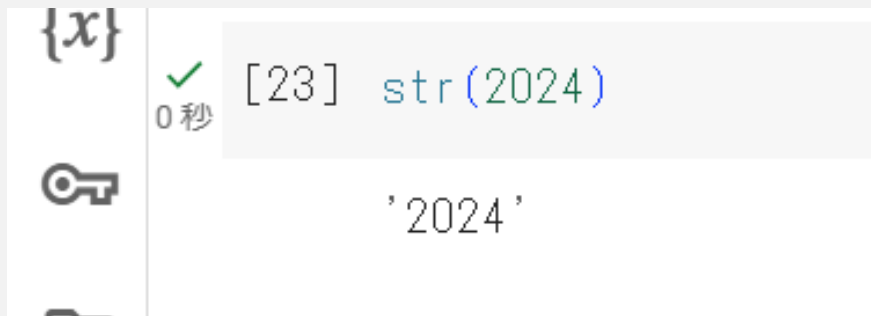
A screenshot of a Python REPL (Jupyter Notebook) showing two code cells. The first cell, labeled [21], contains the code `int("123")` and the output is `123`. The second cell, labeled [22], contains the code `float("123.4")` and the output is `123.4`. The interface includes a left sidebar with icons for running code, navigating, and viewing the output, and a right sidebar for the output area.

```
[21] int("123")
123

[22] float("123.4")
123.4
```

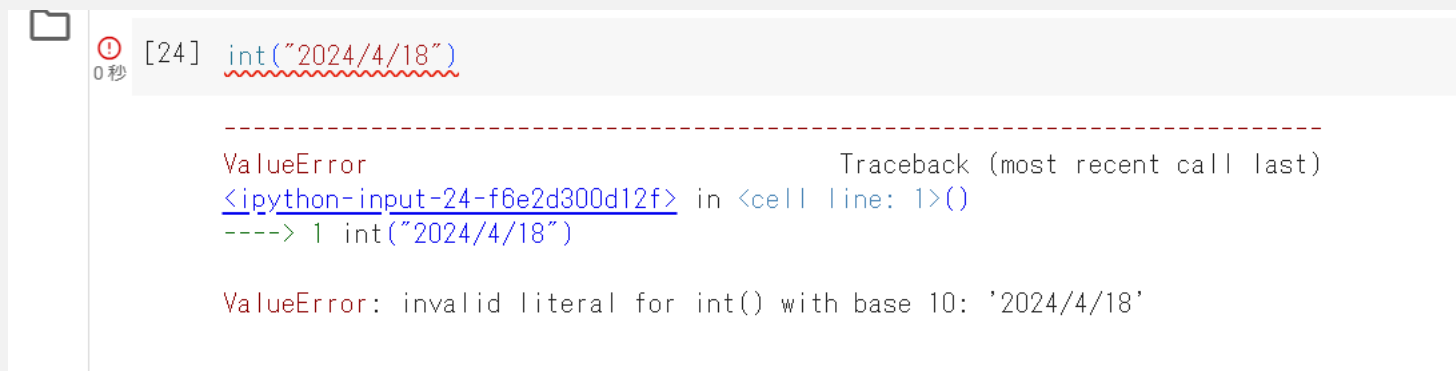
# 文字列と数値の相互変換

逆に数値を数字からなる文字列に変換したい時は`str()`関数を使う。



```
{x}
✓ [23] str(2024)
0秒
'2024'
```

ただし、数字以外の文字を含む文字列を数値に変換することは出来ないので注意



```
[24] int("2024/4/18")
0秒

-----
ValueError                                Traceback (most recent call last)
<ipython-input-24-f6e2d300d12f> in <cell line: 1>()
----> 1 int("2024/4/18")

ValueError: invalid literal for int() with base 10: '2024/4/18'
```

# print関数

今後も比較的頻繁に使うことになる関数がprint関数である。

print関数は引数に指定した値を表示するだけのものであるが、プログラム実行時の変数の値を表示させたい時には必要になる。

引数にはもちろん変数を表示させることが出来るし、複数の値を指定することが出来る。

✓  
0秒

```
[25] print(2024, 4, 18)
```

```
2024 4 18
```

```
[28] print(2024, "/", 4, "/", 18)
```

```
2024 / 4 / 18
```

<>

≡

✓  
0秒

```
[29] num = 10  
      num = num * 10  
      print(num, num * 10)
```

```
100 1000
```



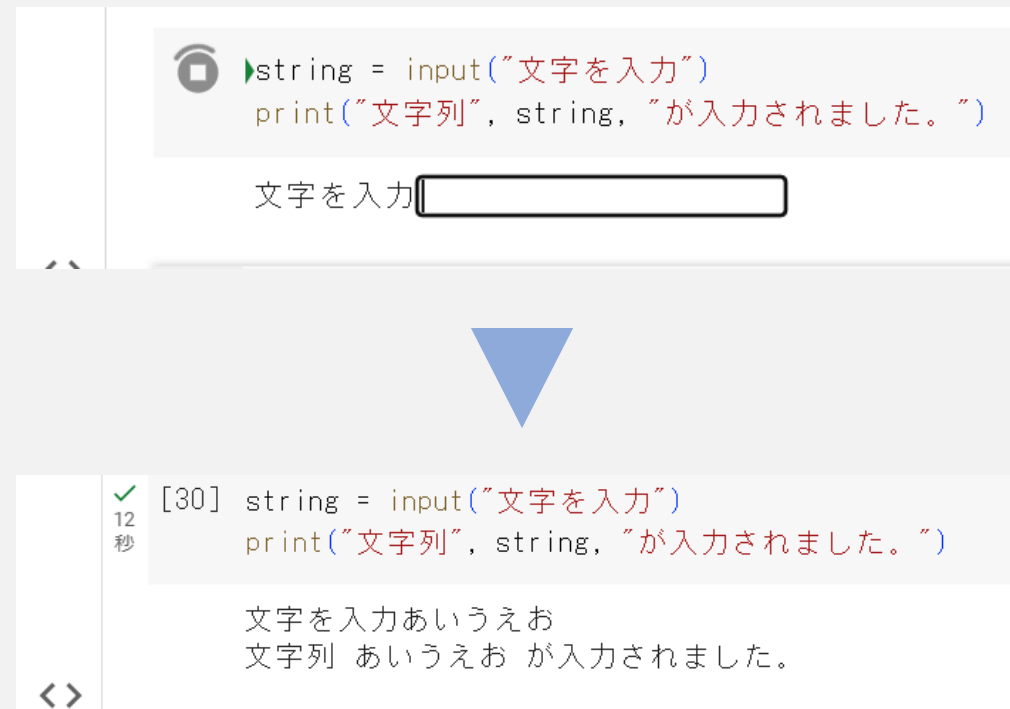
# input関数

input関数は文字列を入力するフィールドを用意し、入力した文字を返すという機能を持つ。

例えば

```
string = input("文字を入力")  
print("文字列", string, "が入力されました。")
```

とすると結果が右の通りになる



```
string = input("文字を入力")  
print("文字列", string, "が入力されました。")
```

文字を入力

▼

```
[30] string = input("文字を入力")  
print("文字列", string, "が入力されました。")
```

文字を入力あいうえお  
文字列 あいうえお が入力されました。

# メソッド

データ型にはそのデータ型特有の操作をする必要があると先のスライドで触れた。

→データ型固有の関数というものがある。これをメソッドという。


例えば、文字データには、文字列全ての英字を大文字に変換するメソッドupper()が存在する。

メソッドはデータ.メソッド名という記法により利用できる。

# メソッド

文字データにはupperメソッド以外にもfindメソッドがある。

文字列textに対してfindメソッドを利用した場合、引数で指定した文字列stringがtextに含まれる場合、textの先頭から見て何番目に文字列stringが含まれているかを返すメソッドである。

A screenshot of a Python REPL (Read-Eval-Print Loop) window. On the left, there is a folder icon and a green checkmark with the text '0秒' (0 seconds). The main area shows the following code: 

```
[1] string = "This is a pen."  
    string.find("pe")
```

 Below the code, the output '10' is displayed. The code is color-coded: 'This is a pen.' is in red, 'pe' is in blue, and '10' is in black.

```
[1] string = "This is a pen."  
    string.find("pe")  
  
10
```

## メソッドと関数の違い

関数とメソッドはかなり似ているが、以下のような違いがあるとされる。

- メソッドはデータ型に固有であるが、関数は必ずしもデータ型によらない。

関数`abs()`は浮動小数点も整数も引数として指定出来た。また`print`関数については数値データに加えて文字データを引数として指定出来る。

一方でメソッドはそうもいかない。