

The Kyoto College of
Graduate Studies
for Informatics

kcg.edu

コンピュータプログラミング概論 (a)

Fundamentals of Computer Programming

2021年秋第2回 eラーニング資料

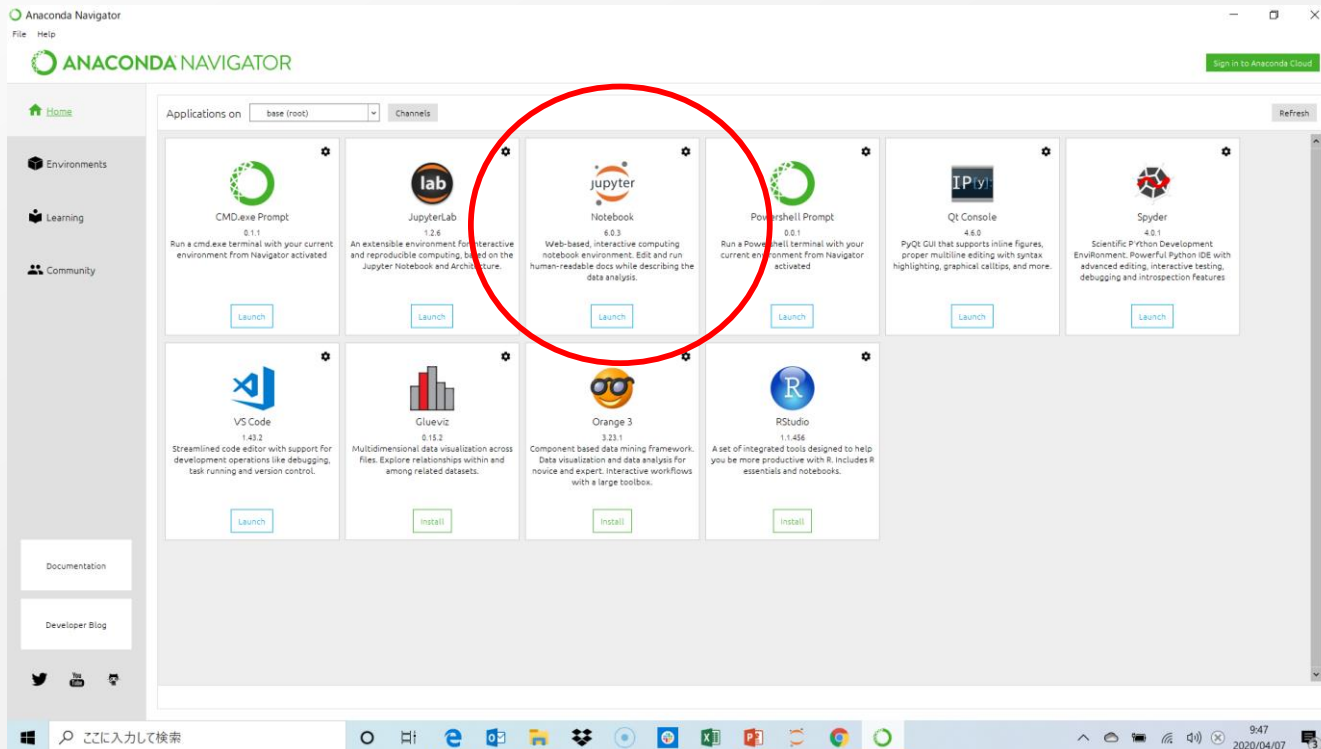
安 平勲

h_an@kcg.ac.jp

PythonをJupyter Notebook（会話型）で動作

Jupyter Notebookでプログラミング

Jupyter Notebookを起動（以下はWindows）



✓ Jupyter NotebookはWebブラウザで動く

Select items to perform actions on them.

Upload New

0 /

	Name ↓	Last Modified	File size
<input type="checkbox"/>	3D Objects	7日前	
<input type="checkbox"/>	anaconda3		
<input type="checkbox"/>	Contacts		
<input checked="" type="checkbox"/>	Desktop	12分前	
<input type="checkbox"/>	Doctor Web	12日前	
<input type="checkbox"/>	Documents	6日前	
<input type="checkbox"/>	Downloads	4日前	
<input type="checkbox"/>	Favorites	24日前	
<input type="checkbox"/>	Links	24日前	
<input type="checkbox"/>	Music	24日前	
<input type="checkbox"/>	OneDrive	4日前	
<input type="checkbox"/>	Pictures	24日前	
<input type="checkbox"/>	Saved Games	24日前	
<input type="checkbox"/>	Searches	24日前	
<input type="checkbox"/>	Videos	9日前	

②新規のJupyter ファイルを作成する

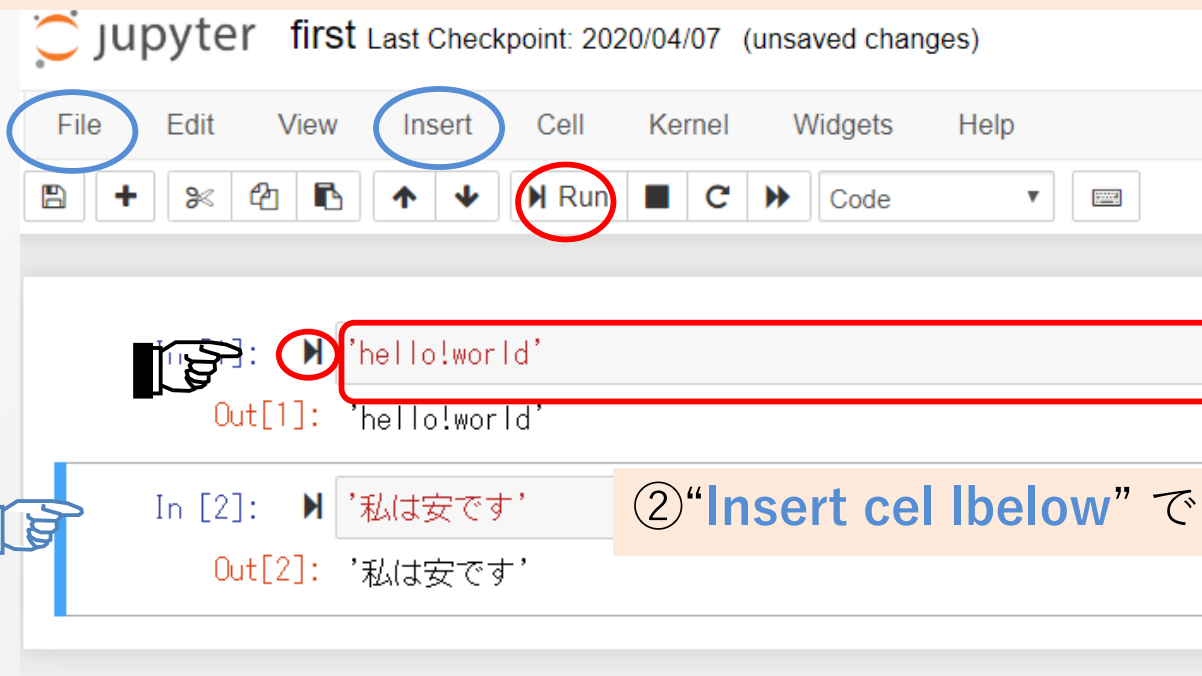
①ファイルを作成したいフォルダを選ぶ

ここに入力して検索

9:24
2020/04/13

Jupyter Notebookでプログラミング

①Jupyterのセルにソースコードを入力。そして、実行



②“Insert cell below”でセルを追加

③次に、 Notebookファイルを保存して、再編集しよう

Jupyter Notebookでプログラミング

Jupyter Notebook でプログラミング

文字列を印字

In [1]: `'こんにちは!私は安です'`
Out[1]: `'こんにちは!私は安です'`

四則演算も同様

In [2]: `10 + 3`
Out[2]: `13`

In [3]: `10 - 3`
Out[3]: `7`

In [4]: `10 * 3`
Out[4]: `30`

In [5]: `10 / 3`
Out[5]: `3.3333333333333335`



✓ `Markdown` を選ぶと、セルに、（コードではなく）コメントが書ける



第一回の四則演算をJupyterで試そう

Jupyter Notebookでのprint()関数

```
In [1]: ▶ 'こんにちは！私は安です'
```

```
Out[1]: 'こんにちは！私は安です'
```

```
In [2]: ▶ 'こんにちは！ 私は安です'
```

```
Out[2]: 'こんにちは！¥u3000私は安です'
```

✓ '¥u3000'は全角空白を意味するUnicode
注) Macでは¥はバックスラッシュ

```
In [3]: ▶ print('こんにちは！ 私は安です')
```

```
こんにちは！ 私は安です
```

 明示的に文字列を出力する場合，print関数を使う

数値, 文字列, 演算子, 変数, ブール値, 配列

Pythonプログラムの構成

■ プログラム言語Pythonを自然言語と対比させると…

プログラム言語；Python

数値，文字列，演算子，変数

文(code)，命令文(statement)

関数，メソッド，クラス

モジュール

パッケージ，ライブラリ

自然言語

名詞，動詞，形容詞などの品詞

文，文章

段落

節，章

論文，本

■ データには型がある

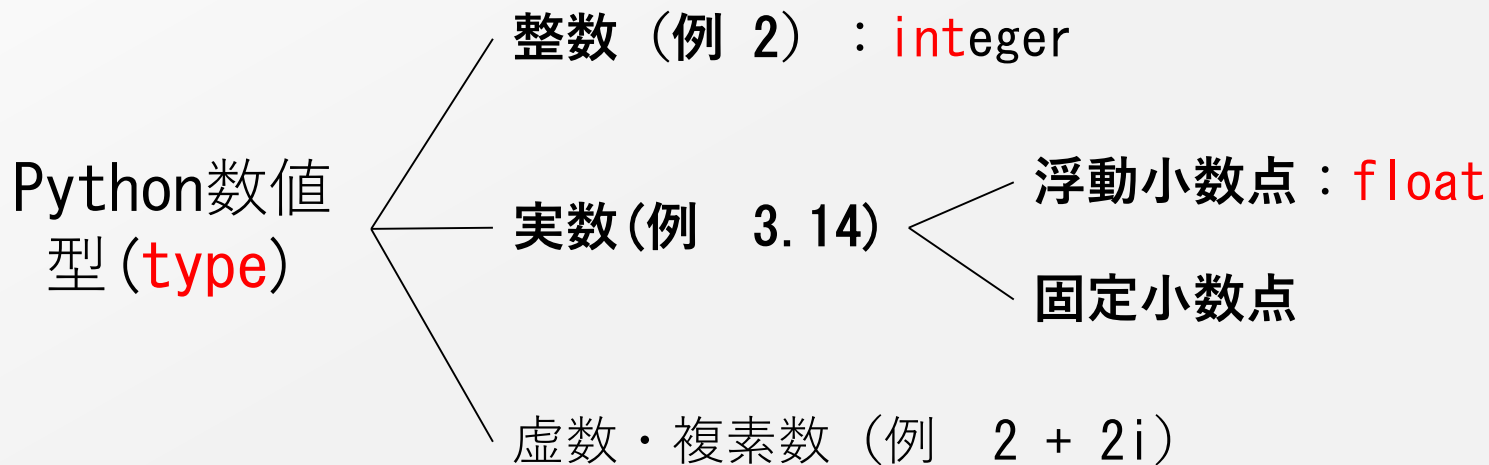
Pythonのデータ型 (data
type)

文字列(string)		“kcgi”, '京都', “123” など (“ または ' で囲む)
数値	整数 (integer)	123, -3, 0 など
	浮動小数点(float)	123.0, 3.14, 0.0など
真偽値(boolean)		True または False

※ 数値の **123** と文字列の **"123"**は別物

データ型typeと数値

- Pythonは**数値**のデータ型として整数，実数，虚数をサポート



虚数はこの講座では勉強しません

算術演算子

算術演算子	意味	例	結果
+	加算	$10 + 3$	13
-	減算	$10 - 3$	7
*	乗算	$10 * 3$	30
/	浮動小数点の除算	$10 / 3$	3.3333
//	整数の除算	$10 // 3$	3
%	剰余	$10 \% 3$	1
**	指数 (べき乗)	$10 ** 3$	1000

浮動小数点（実数）と整数

実数 (float) の演算

```
▶ 10.0 + 3    #小数点を明示すると実数 (float) 演算
8]: 13.0

▶ 10.0 // 3    #小数点を明示すると実数 (float) 演算
9]: 3.0

▶ 10.0 % 3     #余りも実数
10]: 1.0
```

整数 (integer) の演算

```
In [14]: ▶ 10 / 3    #割り算の結果は実数 (float)
Out[14]: 3.3333333333333335

In [15]: ▶ 10 // 3   #割り算の結果は整数 (integer)
Out[15]: 3

In [16]: ▶ 10 % 3    #余りは整数 (integer)
Out[16]: 1

In [17]: ▶ 10 ** 3   #**はべき乗
Out[17]: 1000
```

(参考) 浮動小数点と固定小数点

実数

```
In [29]: 0.099 + 12.001
```

```
Out[29]: 12.1
```

0.099を指数表記すると

```
In [30]: 9.9e-2
```

```
Out[30]: 0.099
```

仮数e指数 : 9.9 (仮数) × 10のマイナス2乗 (指数) ⇒ 小数点の位置が移動 (浮動小数点)

```
In [31]: -9.9e4
```

```
Out[31]: -99000.0
```

```
In [32]: 9.9e-4
```

```
Out[32]: 0.00099
```

※ **0.099**や**0.00099**は固定小数点形式
9.9 e⁻² と**9.9 e⁻⁴**は浮動小数点形式

変数と代入

- 変数 (variable) はプログラムの基本。以下の a が変数

```
In [1]: ▶ a = 10  
        a = a + 3  
        print(a)
```

13

- ① 10を変数aに代入
- ② 変数aに3を加算し、結果をaに代入
- ③ 変数aの内容を出力

✓ print関数は変数も引数にできる



プログラムでは‘=’は等号ではなく、**代入**を意味する。
情報処理試験の疑似言語では‘←’で代入を表現

変数

- 変数 (variable) はデータの入れ物

a = 10



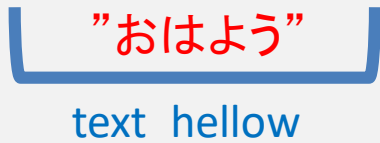
a = a + 3



変数名「a」に数値10を代入(設定ともいう)

「a」に数値3を加算。結果を「a」に代入
⇒ 「a」は13に

text_hellow = "おはよう"



文字列も変数に代入できる

変数名

- 変数の名前ルール(naming rule)
 1. 小文字と大文字の半角英字 (a/Aからz/Z)
 2. 数字 (0から9)
 3. アンダースコア (_)
 4. 小文字と大文字は**区別**される (別の変数とみなされる)
 5. **全角**文字は変数には使えない! (特に空白に注意)
 6. **数字**は名前の**先頭**には使えない!
 7. Pythonの**予約語**は使えない!

✓ naming rule に従わないとプログラムエラーとなる

変数の名前ルール

```
▶ a = 10
  a = A + 1 # 大文字と小文字は別
  print(a)
```


✓ **#** は**コメント文**。説明文などを自由に書ける
注) コメント文は実行されない。無視される

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-1-c85437e86049> in <module>
      1 a = 10
----> 2 a = A + 1 # 大文字と小文字は別
      3 print(a)

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

```
▶ 1a = 10 # 数字で始めるとエラー
  1a = 1a * 10
  print(1a)
```

```
File "<ipython-input-2-c65da70c6e17>", line 1
  1a = 10 # 数字で始めるとエラー
    ^
SyntaxError: invalid syntax
```

 この 3 行のプログラムをjupyterに入力し、修正してみよう

変数の名前ルール：Pythonの予約語

- Pythonの予約語は変数に使えない

```
In [1]: ▶ import keyword  
keyword.kwlist
```

```
Out[1]: ['False',  
         'None',  
         'True',  
         'and',  
         'as',  
         'assert',  
         'async',  
         'await',  
         'break',  
         'class',  
         'continue',  
         'def',  
         'del',  
         'elif',  
         'else',  
         'except',  
         'finally',  
         'for',
```

```
         'for',  
         'from',  
         'global',  
         'if',  
         'import',  
         'in',  
         'is',  
         'lambda',  
         'nonlocal',  
         'not',  
         'or',  
         'pass',  
         'raise',  
         'return',  
         'try',  
         'while',  
         'with',  
         'yield']
```

変数名

- 他人が見ても分かりやすい変数名をつける

自分のBMIを計算してみよう

```
▶ # 自分のBMIを計算
height = 1.70           # 身長
weight = 68             # 体重
bmi = weight / height ** 2 # 計算式
print(bmi)              # 25以上は肥満
```

23.529411764705884

複数同時

セミコロン ; で、複数の代入文が一行に書ける

```
▶ a = 100; b = 1.2345; c = 'Python'  
   print(a,b,c)
```

100 1.2345 Python

✓ print関数の引数には、カンマで複数の
変数、文字列、数値を書ける

一行で複数の変数に代入できる

```
▶ a, b, c = 100, 1.2345, 'Python'  
   print(a,b,c)
```

100 1.2345 Python

文字列 (string) と連結

- シングルクォート'で括る； '文字列'
- ダブルクォート"で括る； "文字列"

シングルクォートの文字列

```
In [1]: ▶ my_familyname = '安'
```

ダブルクォートの文字列

```
In [2]: ▶ my_firstname = "平勲"
```

```
In [3]: ▶ my_name = my_familyname + my_firstname # 文字列の連結
```

```
In [4]: ▶ print(my_name)
```

安平勲

✓ 文字列は + で連結できる

文字列 (string)

■ シングルクォート' とダブルクォート" の使い分け

英語を使う場合は特に注意が必要

```
In [2]: ▶ text1 = " I'm a student of KCGI"  
text2 = " 私は'KCGI'の学生です"  
text3 = " 私は' KCGI' の学生です"  
print(text1,text2,text3)
```

I'm a student of KCGI 私は'KCGI'の学生です 私は' KCGI' の学生です

文字列と連結

■ 数を + すると...

数字を加算

```
In [13]: ▶ number = 10 + 3      # 数字を加算  
          print(number)         # 型を確認  
          type(number)
```

13

Out[13]: int

文字列を連結

```
In [14]: ▶ string = '10' + '3'  # 文字列を連結  
          print(string)         # 結果を確認  
          type(string)         # 型を確認
```

103

Out[14]: str

✓ 数字を ' ' で括ると文字列
なので、「」

文字列；複数行

- 'または"を3つ続けると複数行の文字列が書ける

```
In [11]: a = '''私の名前は  
安  
平麿'''
```

```
In [12]: print(a)  
  
私の名前は  
安  
平麿
```


```
In [13]: a  
  
Out[13]: '私の名前は\n  安\n  平麿 '
```

文字列 ; エスケープシーケンス

■ エスケープ文字 (エスケープシーケンス)

エスケープ文字	意味
¥n	改行
¥t	tab
¥r	キャリッジリターン
¥'	シングルクォート
¥"	ダブルクォート

- '¥'はwindows
- Macでは'¥'
(バックスラッシュ)

 print関数では'¥'の直後の文字を『特殊文字』として扱う

input関数

- **input()**；キーボードから入力した**文字列**を受け取る**関数**

```
strings = input("プロンプト文字列")
```

↑
文字列を受け取る変数

↑
コンソールに表示される文字列

- － 実行すると、コンソールに**プロンプト文字列**を表示して、キーボードからの**入力待つ**状態になる
- － キーボードからenterするまでに**入力した文字列をプログラムに取り込む**
- － 取り込んだ文字列は変数（上の例ではstrings）に代入できる

(参考) Pythonは動的型付け言語

- プログラミング言語は**静的型付け**と**動的型付け**の二種類に大別できる
- 動的型付け言語では、**変数**の型（数値，文字列など）を宣言する必要がない
 - ✓ 静的型付け言語ではプログラマが型宣言する必要あり
 - ✓ Pythonインタプリタが自動で変数の型を判断する

静的型付け 対 動的型付け

静的型付け言語	動的型付け言語
C, C++, C# Java, . . .	Python JavaScript, Perl . .

■ 動的型付け言語のメリット

- a. 記述量（コーディング量）が減る
- b. 比較的簡単にプログラムが書ける ⇒ 生産性が良い

■ 静的型付け言語のメリット

- a. 文法エラーがコンパイル時に分かる
- b. メモリ領域の最適化が図られる ⇒ 処理パフォーマンスが良い

動的型付け

整数 (int) と実数 (float)

```
In [25]: ► integer_number = 10  
float_number = 3.14
```

```
In [26]: ► type(integer_number)
```

```
Out[26]: int
```

```
In [27]: ► type(float_number)
```

```
Out[27]: float
```

整数 (int) と実数 (float)を掛けると

```
In [28]: ► XXX = float_number * integer_number  
print(XXX)
```

```
31.400000000000002
```

```
In [29]: ► type(XXX)
```

```
Out[29]: float
```

Pythonが 'integer_number' と
'float_number' を動的型付け



変数の型は**type()**関数で分かる

Pythonが自動的に判断して
floatに型付け

型変換（キャスト）

- 数値と文字列の加算はできない。型変換が必要

ダメなケース（数値と文字列の加算）

```
In [12]: ▶ string10 = '10'  
          number3 = 3  
          print(string10 + number3)
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-12-5ff31c67b727> in <module>  
      1 string10 = '10'  
      2 number3 = 3  
----> 3 print(string10 + number3)  
  
TypeError: can only concatenate str (not "int") to str
```

文字列型を浮動小数点型に変換する

```
In [13]: ▶ number10 = float(string10)  
          print(number10 + number3)  
  
13.0
```

➤ float関数で実数に型変換
注) int()で整数に型変換

ブール値(boolean) と比較演算子

- **ブール値**は真／偽, Yes／Noなどの2値'**True**'と'**False**'だけを取る型
- 比較演算子は2つの数値, 文字列, 変数を比べ, 下記のブール値を返す

比較演算子	例	説明
==	a == b	a とbが等しい時 True , 等しくなければ False
!=	a != b	a とbが等しくない時True
>	a > b	a がbより大きい時True
>=	a >= b	a がbより大きいか等しい時True
<	a < b	a がbより小さい時True
<=	a <= b	a がbより小さいか等しい時True

ブール値と比較演算子

```
In [14]: ▶ a = 10 ; b = 10 ; c = 3
```

```
In [15]: ▶ a == b
```

```
Out[15]: True
```

```
In [16]: ▶ a != b
```

```
Out[16]: False
```

```
In [17]: ▶ a >= b
```

```
Out[17]: True
```

```
In [18]: ▶ a > c
```

```
Out[18]: True
```

```
In [19]: ▶ a <= c
```

```
Out[19]: False
```

➤ ';'で区切れば，複数命令を1行に書ける

➤ aもbも10なので・・・

➤ aが10, cが3なので・・・

文字列の検索（**in**）とブール値

- 文字列の検索（**in**）でもブール値を返す

```
In [1]: strings = "abcde"  
"a" in strings
```

```
Out[1]: True
```

```
In [2]: "f" in strings
```

```
Out[2]: False
```

➤ 文字列"abcde"に"a"があるのでTrue

➤ "f"はないからFalse

配列とリスト

- 配列 (array) はプログラミングでよく使われるデータ構造
- Pythonでは**リスト**listと呼ぶ組込み (built_in) のデータ型

リスト (list)

```
In [1]: numbers = [2, 4, 6, 8, 10]
        print(numbers)
```

```
[2, 4, 6, 8, 10]
```

```
In [4]: weekdays = ['月曜', '火曜', '水曜', '木曜', '金曜']
        print weekdays
```

```
['月曜', '火曜', '水曜', '木曜', '金曜']
```

リスト (list) のスライスslice

```
In [2]: numbers[0]
```

```
Out[2]: 2
```

```
In [6]: weekdays[0]
```

```
Out[6]: '月曜'
```

```
In [5]: weekdays[4]
```

```
Out[5]: '金曜'
```

- 文字列や数値 (要素と呼ぶ) をカンマ (,) で並べ, 鍵角括弧 **[]** で括る
- リストも変数名をつけられる

- リスト名 **[インデックス番号]** で要素を取り出せる
- インデックス番号は0から始まる要素の順番

リストの連結・検索

リストの連結 (+)


```
▶ weekdays = ['月曜', '火曜', '水曜', '木曜', '金曜']  
weekends = ['土曜', '日曜']  
weeks = weekdays + weekends  
print(weeks)  
print(weeks[0], weeks[6])  
  
['月曜', '火曜', '水曜', '木曜', '金曜', '土曜', '日曜']  
月曜 日曜
```

➤ リストは文字列と同様、連結できる

リストの検索 (in)

```
▶ '土曜' in weeks  
9]: True  
  
▶ '土曜' in weekdays  
1]: False
```

➤ 文字列と同様、検索 (in) もできる

 リスト (list) は別の回で詳しくやります