Jupyter notebook 入門

このノートブックは、Jupyterのいくつかのコア機能の概要を説明しています。

ノートブックでPythonコードを実行する方法に慣れていない場合は、最初にこのノートブックを読む必要があります。

ノートブックの使用に既に慣れている場合は、このコンテンツをスキップするか、参照として保持できます。

このノートでは：

- コードセルとマークダウンセル
- 実行中のコード
- インターフェースの概要
- いくつかの便利なショートカット
- マジックコマンド

New Section

コードセルとマークダウンセル

ノートブックは一連のセルで構成されています。 細胞にはさまざまな種類があります。

コードセルには実行するコードが含まれ、マークダウンセルにはマークダウン構文を使用して記述された説明（ドキュメント）が含まれます。

以下はコードセルです。

- セルの中央をクリックしてアクティブにします
- ツールバーの「実行(Run)」アイコンをクリックしてコードを実行します

In [ ]:
```
# コードセル

# 「#」で始まる行はコメントです。
# 実行されない行です。コードの説明

a = 500
b = 202

print(a + b)
```

```
702
```

代わりに、これは**マークダウン(Markdown)** セルです。

テキストの中央をダブルクリックすると、実際のマークダウンコードが表示されます。

マークダウンコードを見栄えの良いドキュメントにレンダリングするには、セルを実行する必要があります。

マークダウンを使用すると、次のようなものを含むドキュメントをフォーマットできます

大きなヘッダー

小さいヘッダー

はるかに小さいヘッダー

箇条書き：

- One
- Two
- Three

番号付きリスト：

1. One
2. Two
3. Three

italic 、**bold** 、 `monospace` のテキスト。

セルをコードまたはマークダウンとして定義する

ツールバーのドロップダウンメニューを使用してタイプを変更できます。

アクティブセルに応じて、現在のセルタイプがわかります（たとえば、この特定のセルのマークダウン、下のセルのコード）

In [ ]:
```
# コードセル
print("テキスト  JupyterNotebook です。")
```

テキスト　JupyterNotebook です。

コードを実行する

コードセルを実行するには、いくつかのオプションがあります。

- ツールバーの「実行」アイコンをクリックします
- [セル]メニュー、[セルを実行]（または他の[実行...]オプションのいずれか）をクリックします。
- Shift + Enterを使用します
- alt + enterを使用します（これによりアクティブセルが実行され、すぐ下に空のセルが追加されます）

通知

実行の順序は重要です。ノートブックは上から下に実行する必要があります。

ノートブック環境はステートフル(全てのセル状態を記憶)です。セルを実行した後、その状態は次のセルで使用できます（下の例を参照）

In [ ]:
```
var = "Python Jupyter Notebook"

print("この {} ツールを勉強している.".format(var))
```

この Python Jupyter Notebook ツールを勉強している.

In [ ]:
```
# 上記のセルを実行した場合、変数varは引き続き使用できます
print(var)
```

Python Jupyter Notebook

コードセルを実行すると、最後の式の出力がデフォルトで出力され、 `print()` 明示的にする必要はありません。 :

In [ ]:
```
var
```

Out[ ]: 'Python Jupyter Notebook'

In [ ]:
```
a = 500
b = 400

a + b
```

Out[ ]: 900

インターフェースの概要

各コードセルの横に、実行順序のプレースホルダーがあります。これにより、実行されたセルと順序がわかります。

例

次のセルを実行し、クリックして再度実行します。

増加する数を観察します。

In [ ]:
```
print("これを複数実行する")
```

これを複数実行する

主な機能

[編集]メニューを使用すると、コピー/貼り付けやマージなどのセルの操作を実行できます。

「表示」メニューでは、ヘッダー、ツールバー、および行番号を視覚化または非表示にできます。

「挿入」メニューでは、新しい空のセルを挿入できます。

「セル」メニューでは、さまざまな方法でセルを実行し、セルタイプを変更して、出力をクリアできます。

「カーネル」メニューでは、カーネル（ノートブックの背後にあるPythonインタープリター）と対話できます。

ヒント

コードセルが動かなくなった場合（無限ループなど）、カーネルを**中断**または**再起動**する必要がある場合があります。

いくつかの便利なショートカット

キーボードショートカットを使用すると、生産性や快適性が向上する場合は、十分な数があります。

ショートカットを使用できるようにするには、[Esc]キーを押して「コマンドモード」に切り替える必要があります。

`Esc` を押すと、アクティブなコードセルは（緑の境界線ではなく）青い境界線を表示し、キーボードコマンドを受け入れることができます。

「編集モード」に戻るには、「Enter」を押すか、セルの中央をクリックします（セルの境界線が緑色に変わります）。

- `Esc+M` : セルタイプをMarkdownに変更する

- Esc+Y : セルタイプをコードに変更する

- Esc + A ：上に新しい空のセルを追加
- Esc + B ：新しい空のセルを下に追加
- Esc + DD （2つのD）：現在のセルを削除

- Esc + X ：選択したセルを切り取ります
- Esc + C ：選択したセルをコピーします
- Esc + V ：選択したセルを貼り付け

さらに多くのコマンドとショートカットが利用可能で、以下を使用してリストを視覚化できます。

- Esc + P ：コマンドパレットを表示

Jupyter Magics

マジック（またはマジックコマンド）は、Jupyterノートブックを強化する便利なヘルパー機能です。

コードセルは魔法のコマンドを実行できます。

マジックには2つのタイプがあります。ラインマジック（単一の % で始まる）とセルマジック（二重の %% で始まる）です。

利用可能なものの完全なリストを参照してください：

In [3]:
```
%lsmagic
```

Out[3]:
```
Available line magics:
%alias  %alias_magic  %autoawait  %autocall  %automagic  %autosave  %bookmark  %cd  %clear  %cls  %colors  %conda  %config  %connect_inf
o  %copy  %ddir  %debug  %dhist  %dirs  %doctest_mode  %echo  %ed  %edit  %env  %gui  %hist  %history  %killbgscripts  %ldir  %less  %lo
ad  %load_ext  %loadpy  %logoff  %logon  %logstart  %logstate  %logstop  %ls  %lsmagic  %macro  %magic  %matplotlib  %mkdir  %more  %not
ebook  %page  %pastebin  %pdb  %pdef  %pdoc  %pfile  %pinfo  %pinfo2  %pip  %popd  %pprint  %precision  %prun  %psearch  %psource  %push
d  %pwd  %pycat  %pylab  %qtconsole  %quickref  %recall  %rehashx  %reload_ext  %ren  %rep  %rerun  %reset  %reset_selective  %rmdir  %r
un  %save  %sc  %set_env  %store  %sx  %system  %tb  %time  %timeit  %unalias  %unload_ext  %who  %who_ls  %whos  %xdel  %xmode

Available cell magics:
%%!  %%HTML  %%SVG  %%bash  %%capture  %%cmd  %%debug  %%file  %%html  %%javascript  %%js  %%latex  %%markdown  %%perl  %%prun  %%pypy
%%python  %%python2  %%python3  %%ruby  %%script  %%sh  %%svg  %%sx  %%system  %%time  %%timeit  %%writefile

Automagic is ON, % prefix IS NOT needed for line magics.
```

実際に役立つマジック

test.pyを作成してから、

```
#test.py
print("Hello World!")
```

以下のセルを実行して、外部ファイルからコードをロードします

In [ ]:
```
# %load ./test.py
#test.py
print("Hello World!")
```

```
Hello World!
```

以下のセルを実行して、test.pyファイルからコードを実行します

In [1]:
```
%%bash
echo "print('Hello Python!')" > test.py
```

```
Couldn't find program: 'bash'
```

In [2]:
```
%cat "print('Hello Python!')" > test.py
```

```
UsageError: Line magic function `%cat` not found.
```

In [ ]:
```
%run ./test.py
```

```
ERROR:root:File `'./test.py'` not found.
```

以下の2つのセルを実行して、コードの時間を計ります。

注：これはセルマジックの例です。

In [ ]:
```
%%time

slow_sum = 0

for i in range(1000000):
    slow_sum = slow_sum + i

slow_sum
```

```
CPU times: user 145 ms, sys: 0 ns, total: 145 ms
Wall time: 150 ms
```

In [ ]:
```
%%time

faster_sum = sum(range(1000000))
```

```
faster_sum
```

```
CPU times: user 27.1 ms, sys: 949 µs, total: 28.1 ms
Wall time: 29.2 ms
```

Pythonの基本

- 基本データ型
- 式と代入文

- 式

In [ ]:
```
1729
```

Out[ ]: 1729

In [4]:
```
12**3+1**3
```

Out[4]: 1729

In [5]:
```
'知能'+'システム'
```

Out[5]: '知能システム'

In [6]:
```
'すも'*5
```

Out[6]: 'すもすもすもすもすも'

In [7]:
```
100/3
```

Out[7]: 33.333333333333336

In [8]:
```
100//3
```

Out[8]: 33

In [9]:
```
100 % 3
```

Out[9]: 1

- リテラル：更新不能なデータ

In [ ]:
```
# 浮動小数点数
3.14159265358
```

Out[ ]: 3.14159265358

In [39]:
```
[('Celsius','Fahreneit'),('Kilometer','Mile'),('Centimeter','Inch'),('Kilogram','Pound'),('Square meter','坪')]
```

Out[39]:
```
[('Celsius', 'Fahreneit'),
 ('Kilometer', 'Mile'),
 ('Centimeter', 'Inch'),
 ('Kilogram', 'Pound'),
 ('Square meter', '坪')]
```

In [1]:
```
123.456e-14
```

Out[1]: 1.23456e-12

In [3]:
```
[0b100,0o100,0x100]
```

Out[3]: [4, 64, 256]

In [ ]:
```
'Pythonへようこそ！'
```

Out[ ]: 'Pythonへようこそ！'

- 代入文

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

In [11]:
```
x
```

Out[11]: 3

In [13]:
```
x += 1000
x
```

Out[13]: 1003

- 型の確認

In [ ]:
```
# 型の確認
type(1+5j) # 複素数型
```

Out[ ]: complex

In [ ]:
```
# 型の確認
type('Pythonへようこそ！') # 文字列型  str
```

Out[ ]: str

In [ ]:
```
type(3.14159265358)
```

Out[ ]: float

- シーケンスのアクセス

In [ ]:
```
# シーケンスのアクセス
for a in 1,'文字',3.14,1+5j:
  print(a)
```

```
1
文字
3.14
(1+5j)
```

In [15]:
```
s = 'すもももももももものうち'
print(s[0],s[1],s[-2],s[-1])
```

```
す も う ち
```

In [17]:
```
print(s[1:-3])
```

```
ももももももも
```

In [19]:
```
s1 = s[:2]+'す'+s[3:]
s1
```

Out[19]: 'すもすももももももものうち'

In [21]:
```
文 = """
Python（パイソン）はインタープリタ型の高水準汎用プログラミング言語である。
グイド・ヴァン・ロッサムにより創り出され、1991年に最初にリリースされたPythonの設計哲学は、
有意なホワイトスペース(オフサイドルール)の顕著な使用によってコードの可読性を重視している。
その言語構成とオブジェクト指向のアプローチは、プログラマが小規模なプロジェクトから大規模な
プロジェクトまで、明確で論理的なコードを書くのを支援することを目的としている。
"""
文
```

Out[21]: '¥nPython（パイソン）はインタープリタ型の高水準汎用プログラミング言語である。¥nグイド・ヴァン・ロッサムにより創り出され、1991年に最初にリリースされたPythonの設計哲学は、¥n有意なホワイトスペース(オフサイドルール)の顕著な使用によってコードの可読性を重視している。¥nその言語構成とオブジェクト指向のアプローチは、プログラマが小規模なプロジェクトから大規模な¥nプロジェクトまで、明確で論理的なコードを書くのを支援することを目的としている。¥n'

In [22]:
```
type(文)
```

Out[22]: str

In [23]:
```
len(文)
```

Out[23]: 226

In [24]:
```
文.count('Python')
```

Out[24]: 2

In [26]:
```
文=文.replace('¥n','')
文
```

Out[26]: 'Python（パイソン）はインタープリタ型の高水準汎用プログラミング言語である。グイド・ヴァン・ロッサムにより創り出され、1991年に最初にリリースされたPythonの設計哲学は、有意なホワイトスペース(オフサイドルール)の顕著な使用によってコードの可読性を重視している。その言語構成とオ

ブジェクト指向のアプローチは、プログラマが小規模なプロジェクトから大規模なプロジェクトまで、明確で論理的なコードを書くのを支援することを目的としている。'

In [27]:
```
文のリスト=文.split('。')
文のリスト
```

Out[27]:
```
['Python（パイソン）はインタープリタ型の高水準汎用プログラミング言語である',
 'グイド・ヴァン・ロッサムにより創り出され、1991年に最初にリリースされたPythonの設計哲学は、有意なホワイトスペース(オフサイドルール)の顕著な使用によってコードの可読性を重視している',
 'その言語構成とオブジェクト指向のアプローチは、プログラマが小規模なプロジェクトから大規模なプロジェクトまで、明確で論理的なコードを書くのを支援することを目的としている',
 '']
```

In [28]:
```
del 文のリスト[-1]
文のリスト
```

Out[28]:
```
['Python（パイソン）はインタープリタ型の高水準汎用プログラミング言語である',
 'グイド・ヴァン・ロッサムにより創り出され、1991年に最初にリリースされたPythonの設計哲学は、有意なホワイトスペース(オフサイドルール)の顕著な使用によってコードの可読性を重視している',
 'その言語構成とオブジェクト指向のアプローチは、プログラマが小規模なプロジェクトから大規模なプロジェクトまで、明確で論理的なコードを書くのを支援することを目的としている']
```

In [29]:
```
文のリスト[-1]
```

Out[29]:
```
'その言語構成とオブジェクト指向のアプローチは、プログラマが小規模なプロジェクトから大規模なプロジェクトまで、明確で論理的なコードを書くのを支援することを目的としている'
```

In [30]:
```
s3 = 文のリスト[-1]
len(s3)
```

Out[30]:
```
83
```

In [31]:
```
s3.split('、')
```

Out[31]:
```
['その言語構成とオブジェクト指向のアプローチは',
 'プログラマが小規模なプロジェクトから大規模なプロジェクトまで',
 '明確で論理的なコードを書くのを支援することを目的としている']
```

- 辞書型

In [9]:
```
色 = {'赤':0xff0000,'緑':0x00ff00,'青':0x0000ff}
色
```

Out[9]:
```
{'赤': 16711680, '緑': 65280, '青': 255}
```

In [11]:
```
RGB = {
    (255,0,0) : '赤',
    (0,255,0) : '緑',
    (0,0,255) : '青'
}
RGB
```

Out[11]:
```
{(255, 0, 0): '赤', (0, 255, 0): '緑', (0, 0, 255): '青'}
```

In [12]:
```
色['赤']
```

Out[12]:
```
16711680
```

In [13]:
```
RGB[(255,0,0)]
```

Out[13]:
```
'赤'
```

In [14]:
```
色.keys()
```

Out[14]:
```
dict_keys(['赤', '緑', '青'])
```

In [15]:
```
RGB.keys()
```

Out[15]:
```
dict_keys([(255, 0, 0), (0, 255, 0), (0, 0, 255)])
```

In [16]:
```
色.values()
```

Out[16]:
```
dict_values([16711680, 65280, 255])
```

In [17]:
```
RGB.values()
```

Out[17]:
```
dict_values(['赤', '緑', '青'])
```

ステートメント

- if 文
- while 文
- for 文
- continue 文
- 例外処理
- assert 文
- with 文

In [41]:
```
temp_unit = 'Celsius'
print(temp_unit)
if temp_unit == 'Fahreneit':
    print("T in   ° F")
elif temp_unit == 'Celsius':
    print("T in ° C")
elif temp_unit =='Kelvin':
    print("T in ° K")
```

```
Celsius
T in ° C
```

In [42]:
```
units = [('Celsius','Fahreneit'),('Kilometer','Mile'),('Centimeter','Inch'),('Kilogram','Pound'),('Square meter','坪')]
i = 0
stop = len(units)
while i<stop:
    print(i)
    k1,k2 = units[i]
    print("unit pair: (",k1,',',k2,')')
    i+=1
```

```
0
unit pair: ( Celsius , Fahreneit )
1
unit pair: ( Kilometer , Mile )
2
unit pair: ( Centimeter , Inch )
3
unit pair: ( Kilogram , Pound )
4
unit pair: ( Square meter , 坪 )
```

In [43]:
```
for k1,k2 in units:
    print("unit pair: (",k1,',',k2,')')
```

```
unit pair: ( Celsius , Fahreneit )
unit pair: ( Kilometer , Mile )
unit pair: ( Centimeter , Inch )
unit pair: ( Kilogram , Pound )
unit pair: ( Square meter , 坪 )
```

In [32]:
```
for 文字 in '知能はシステム':
   if 文字 == 'は':
      continue
   print ('文字 :', 文字)
```

```
文字 : 知
文字 : 能
文字 : シ
文字 : ス
文字 : テ
文字 : ム
```

In [33]:
```
# 例外処理
try:
    1/0
except ZeroDivisionError as exc:
    print(exc)
```

```
division by zero
```

In [36]:
```
# assert文
def KelvinToFahrenheit(Temperature):
   assert (Temperature >= 0),"0° K以下は絶対零度よりも寒い(Colder than absolute zero)!"
   return ((Temperature-273)*1.8)+32

print ('273° Kは',KelvinToFahrenheit(273),'° F')
print ('505.78° Kは',int(KelvinToFahrenheit(505.78)),'° F')
print ('-5° Kは',KelvinToFahrenheit(-5),'° F')
```

```
273° Kは 32.0 ° F
505.78° Kは 451 ° F
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-36-f024ef58f10f> in <module>
      6 print ('273° Kは',KelvinToFahrenheit(273),'° F')
      7 print ('505.78° Kは',int(KelvinToFahrenheit(505.78)),'° F')
----> 8 print ('-5° Kは',KelvinToFahrenheit(-5),'° F')

<ipython-input-36-f024ef58f10f> in KelvinToFahrenheit(Temperature)
      1 # assert文
      2 def KelvinToFahrenheit(Temperature):
----> 3     assert (Temperature >= 0),"0° K以下は絶対零度よりも寒い(Colder than absolute zero)!"
      4     return ((Temperature-273)*1.8)+32
      5

AssertionError: 0° K以下は絶対零度よりも寒い(Colder than absolute zero)!
```

In [37]:
```
# with文
文 = """
Python（パイソン）はインタープリタ型の高水準汎用プログラミング言語である。
グイド・ヴァン・ロッサムにより創り出され、1991年に最初にリリースされたPythonの設計哲学は、
有意なホワイトスペース(オフサイドルール)の顕著な使用によってコードの可読性を重視している。
その言語構成とオブジェクト指向のアプローチは、プログラマが小規模なプロジェクトから大規模な
プロジェクトまで、明確で論理的なコードを書くのを支援することを目的としている。
"""
with open('wiki.txt', 'w') as file:
    file.write(文)
```

In [38]:
```
%ls
```

```
 ドライブ C のボリューム ラベルがありません。
 ボリューム シリアル番号は 061F-51F2 です

 C:¥Users¥david¥Documents¥Practice2021 のディレクトリ

2021/04/20  13:43    <DIR>          .
2021/04/20  13:43    <DIR>          ..
2021/04/20  12:12    <DIR>          .ipynb_checkpoints
2021/04/20  13:42            67,283 Basic20210420.ipynb
2021/04/20  12:06            41,482 Basic20210420_JP.ipynb
2021/04/19  16:01         1,362,832 students-name.docx
2021/04/20  13:43               434 wiki.txt
2021/04/06  12:59               354 コース演習I_2021.txt
               5 個のファイル           1,472,385 バイト
               3 個のディレクトリ  137,647,845,376 バイトの空き領域
```

関数

- def 関数名(引数1,引数2,...): ...

In [ ]:
```
```

Pythonスクリプト

- 関数、変数、型
- リスト
- 文字列型

In [4]:
```
def num_to_str(n):
    return str(n)

def str_to_int(s):
    return int(s)

def str_to_float(f):
    return float(f)

if __name__ == "__main__":
    # コメント：実行されない行
    '''
    コメント：実行されない複数の行
    -------------
    上記の関数を呼び出す
    '''
    float_num = 999.01
    int_num = 87
    float_str = '23.09'
    int_str = '19'
    string = 'how now brown cow'
    s_float = num_to_str(float_num)
    s_int = num_to_str(int_num)
    i_str = str_to_int(int_str)
    f_str = str_to_float(float_str)
    print(s_float, 'は', type(s_float),'型')
    print(s_int, 'は', type(s_int),'型')
    print(f_str, 'は', type(f_str),'型')
    print(i_str, 'は', type(i_str),'型')
    print('¥nstring', '"' + string + '"は', len(string), '文字')
    str_ls = string.split()
    print('文字を分割:', str_ls)
    print('リストを結合:', ' '.join(str_ls))
```

```
999.01 は <class 'str'> 型
87 は <class 'str'> 型
23.09 は <class 'float'> 型
19 は <class 'int'> 型

string "how now brown cow"は 17 文字
文字を分割: ['how', 'now', 'brown', 'cow']
リストを結合: how now brown cow
```

- リスト型 [item1,item2,...]

In [5]:
```
# リスト型[]

if __name__ == "__main__":
    ls = ['orange', 'banana', 10, 'leaf', 77.009, 'tree', 'cat']
    print('リストの要素数:', len(ls), 'items')
```

```
print('catの数:', ls.count('cat'), ',', 'catの位置:', ls.index('cat'))
print('¥nlistの操作:')
cat = ls.pop(6) #リストの6番目を抜き出す
print('cat:', cat, ', list:', ls)
ls.insert(0, 'cat') #リストの0番目に'cat'を挿入
ls.append(99) #リストに99を追加
print(ls)
ls[7] = '11' #リストの7番目を11に変更
print(ls)
ls.pop(1)
print(ls)
ls.pop()
print(ls)
print('¥nlistのスライス:')
print('先頭～３番目まで                    :', ls[:3])
print('末尾～３番目まで                    :', ls[3:])
print('2～5番目                            :', ls[1:5])
print('末尾から３番目～末尾まで            :', ls[-3:])
print('先頭から2番目～末尾から2番目        :', ls[1:-1])
print('¥n別のリストから新しいリストを作成:')
print('list                                :', ls)
fruit = ['orange']
more_fruit = ['apple', 'kiwi', 'pear']
fruit.append(more_fruit)
print('appendしたフルーツリスト            :', fruit)
fruit.pop(1)
fruit.extend(more_fruit)
print('extendしたフルーツリスト            :', fruit)
a, b = fruit[2], fruit[1]
print('sliceしたフルーツ                   :', a, b)
```

```
リストの要素数: 7 items
catの数: 1 , catの位置: 6

listの操作:
cat: cat , list: ['orange', 'banana', 10, 'leaf', 77.009, 'tree']
['cat', 'orange', 'banana', 10, 'leaf', 77.009, 'tree', 99]
['cat', 'orange', 'banana', 10, 'leaf', 77.009, 'tree', '11']
['cat', 'banana', 10, 'leaf', 77.009, 'tree', '11']
['cat', 'banana', 10, 'leaf', 77.009, 'tree']

listのスライス:
先頭～３番目まで                    : ['cat', 'banana', 10]
末尾～３番目まで                    : ['leaf', 77.009, 'tree']
2～5番目                            : ['banana', 10, 'leaf', 77.009]
末尾から３番目～末尾まで            : ['leaf', 77.009, 'tree']
先頭から2番目～末尾から2番目        : ['banana', 10, 'leaf', 77.009]

別のリストから新しいリストを作成:
list                                : ['cat', 'banana', 10, 'leaf', 77.009, 'tree']
appendしたフルーツリスト            : ['orange', ['apple', 'kiwi', 'pear']]
extendしたフルーツリスト            : ['orange', 'apple', 'kiwi', 'pear']
sliceしたフルーツ                   : kiwi apple
```

- タプル型 (item1,item2,...)

In [6]:

```
# タプル型()

if __name__ == "__main__":
    tup = ('orange', 'banana', 'grape', 'apple', 'grape')
    print('tupleの要素数:', len(tup))
    print('grapeの数:', tup.count('grape'))
    print('¥ntupleのスライス:')
    print('先頭～３番目まで                :', tup[:3])
    print('末尾～３番目まで                :', tup[3:])
    print('2～5番目                        :', tup[1:5])
    print('末尾から３番目～末尾まで        :', tup[-3:])
    print('先頭から2番目～末尾から2番目    :', tup[1:-1])
```

```
tupleの要素数: 5
grapeの数: 2

tupleのスライス:
先頭～３番目まで                : ('orange', 'banana', 'grape')
末尾～３番目まで                : ('apple', 'grape')
2～5番目                        : ('banana', 'grape', 'apple', 'grape')
末尾から３番目～末尾まで        : ('grape', 'apple', 'grape')
先頭から2番目～末尾から2番目    : ('banana', 'grape', 'apple')
```

- 辞書型{key1:value1,key2:value2,...}

In [7]:

```
# 辞書型{}

if __name__ == "__main__":
    audio = {'amp':'Linn', 'preamp':'Luxman', 'speakers':'Energy',
             'ic':'Crystal Ultra', 'pc':'JPS', 'power':'Equi-Tech',
             'sp':'Crystal Ultra', 'cdp':'Nagra', 'up':'Esoteric'}
    print("辞書（キー:値）のリスト:¥n",audio)
    print("辞書のキー: ¥n",list(audio.keys()))
    del audio['up']
    print('"up"の要素を削除')
    print(audio, '¥n')
    print('辞書に要素を追加')
    audio['up'] = 'Oppo'
    print(audio, '¥n')
    print('汎用オーディオ機器は:', audio['up'], '¥n')
    dict_ls = [audio] #辞書をリストに格納する
```

```
    video = {'tv':'LG 65C7 OLED', 'stp':'DISH', 'HDMI':'DH Labs',
             'cable' : 'coax'}
    print('辞書オブジェクトのリスト')
    dict_ls.append(video) #  dict_lsリストにvideo辞書を追加する
    for i, row in enumerate(dict_ls):
        print(i,'行目',':')
        print(row)
```

```
辞書（キー:値）のリスト:
 {'amp': 'Linn', 'preamp': 'Luxman', 'speakers': 'Energy', 'ic': 'Crystal Ultra', 'pc': 'JPS', 'power': 'Equi-Tech', 'sp': 'Crystal Ultra', 'cdp': 'Nagra', 'up': 'Esoteric'}
辞書のキー:
 ['amp', 'preamp', 'speakers', 'ic', 'pc', 'power', 'sp', 'cdp', 'up']
"up"の要素を削除
{'amp': 'Linn', 'preamp': 'Luxman', 'speakers': 'Energy', 'ic': 'Crystal Ultra', 'pc': 'JPS', 'power': 'Equi-Tech', 'sp': 'Crystal Ultra', 'cdp': 'Nagra'}

辞書に要素を追加
{'amp': 'Linn', 'preamp': 'Luxman', 'speakers': 'Energy', 'ic': 'Crystal Ultra', 'pc': 'JPS', 'power': 'Equi-Tech', 'sp': 'Crystal Ultra', 'cdp': 'Nagra', 'up': 'Oppo'}

汎用オーディオ機器は: Oppo

辞書オブジェクトのリスト
0 行目 :
{'amp': 'Linn', 'preamp': 'Luxman', 'speakers': 'Energy', 'ic': 'Crystal Ultra', 'pc': 'JPS', 'power': 'Equi-Tech', 'sp': 'Crystal Ultra', 'cdp': 'Nagra', 'up': 'Oppo'}
1 行目 :
{'tv': 'LG 65C7 OLED', 'stp': 'DISH', 'HDMI': 'DH Labs', 'cable': 'coax'}
```

- リスト内包 (List Comprehension)

In [8]:

```
# リスト内包 (List Comprehension)

if __name__ == "__main__":
    miles = [100, 10, 9.5, 1000, 30]
    kilometers = [x * 1.60934 for x in miles]
    print ('milesをkmに変換:')
    for i, row in enumerate(kilometers):
        print ('{:>4} {:>8}{:>8} {:>2}'.
               format(miles[i],'miles is', round(row,2), 'km'))
    print('¥npet:')
    pet = ['cat', 'dog', 'rabbit', 'parrot', 'guinea pig', 'fish']
    print(pet)
    print('¥npets:')
    pets = [x + 's' if x != 'fish' else x for x in pet]
    print(pets)
    subset = [x for x in pets if x != 'fish' and x != 'rabbits'
              and x != 'parrots' and x != 'guinea pigs']
    print('¥n一般的なペット:')
    print(subset[1], 'and', subset[0])
    sales = [9000, 20000, 50000, 100000]
    print('¥nボーナス:')
    bonus = [0 if x < 10000 else x * .02 if x >= 10000 and x <= 20000
             else x * .03 for x in sales]
    print(bonus)
    print('¥ndict型ボーナス:')
    people = ['dave', 'sue', 'al', 'sukki']
    d = {}
    for i, row in enumerate(people):
        d[row] = bonus[i]
    print(d, '¥n')
    print('{:<3} {:<4}'.format('従業員', 'ボーナス'))
    for k, y in d.items():
        print('{:<6} {:>6}'.format(k, y))
```

```
milesをkmに変換:
 100 miles is  160.93 km
  10 miles is   16.09 km
 9.5 miles is   15.29 km
1000 miles is 1609.34 km
  30 miles is   48.28 km

pet:
['cat', 'dog', 'rabbit', 'parrot', 'guinea pig', 'fish']

pets:
['cats', 'dogs', 'rabbits', 'parrots', 'guinea pigs', 'fish']

一般的なペット:
dogs and cats

ボーナス:
[0, 400.0, 1500.0, 3000.0]

dict型ボーナス:
{'dave': 0, 'sue': 400.0, 'al': 1500.0, 'sukki': 3000.0}

従業員 ボーナス
dave        0
sue     400.0
al     1500.0
sukki  3000.0
```

In [ ]: