2018年1月28日
第209回農林交流センターワークショップ
メッシュ農業気象データ利用講習会

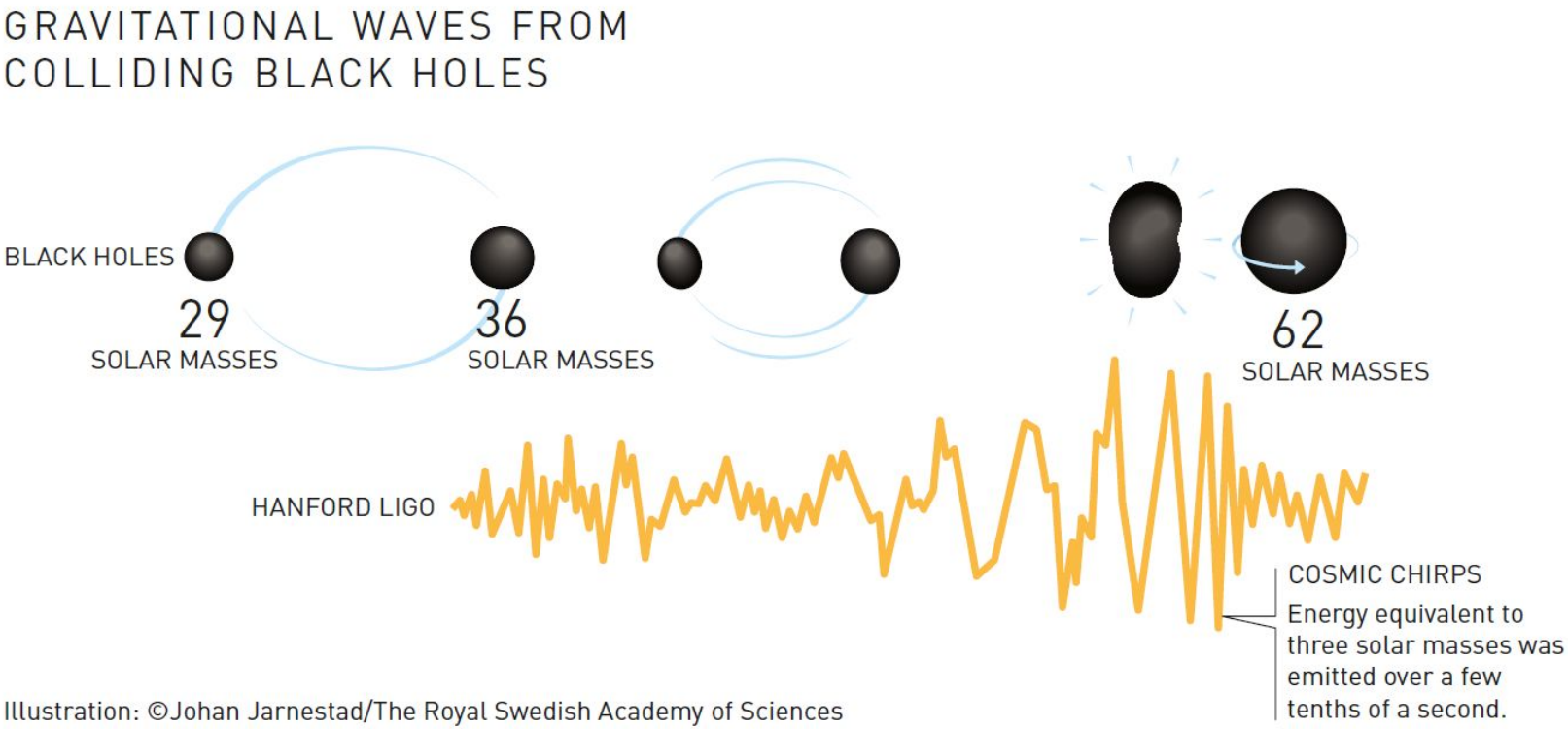
Pythonによる プログラミングの基礎

https://amu.rd.naro.go.jp/wiki_open/doku.php?id=start

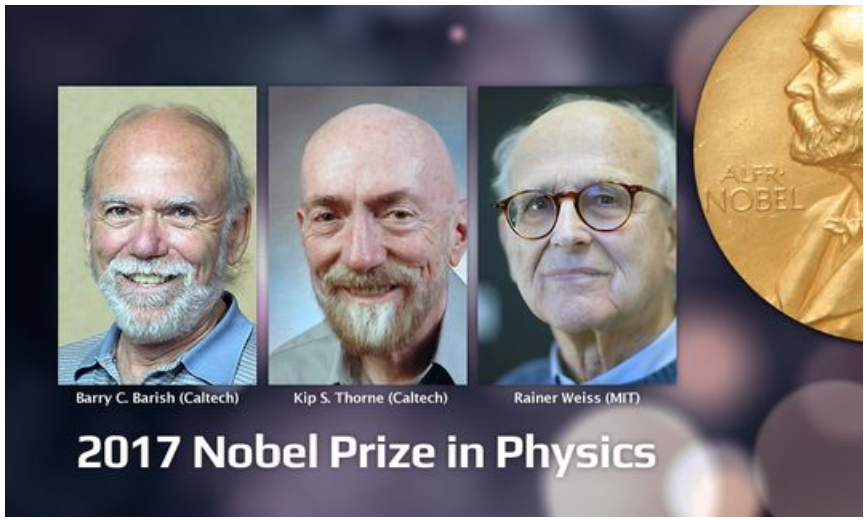
片柳薫子
農研機構　農業環境変動研究センター


```
# 数あるプログラミング言語のひとつ
# 比較的書きやすい
```

```
import this
# the Zen of Python
```



2017年
ノーベル物理学賞



https://www.nobelprize.org/nobel_prizes/physics/laureates/2017/press.html

LIGO Open Science Center

LIGO is operated by California Institute of Technology and Massachusetts Institute of Technology and supported by the U.S. National Science Foundation.

Tutorials

Each tutorial will lead you step-by-step through some common data analysis tasks. While LIGO data can be analyzed using libraries in many software languages (C, C++, Matlab, etc.), most of these tutorials use Python. See also the software examples page for more examples.

See the software setup page for help installing software to run these tutorials.

most of these tutorials use Python

Binary Black Hole Events

Use matched filtering to find signals hidden in noise.

Run: Azure | mybinder (Beta)

View: GW150914 | LVT151012 | GW151226 | GW170104

Download: zip file with data | Jupyter notebook | python script

LIGO = 重力波望遠鏡

https://losc.ligo.org/tutorials/

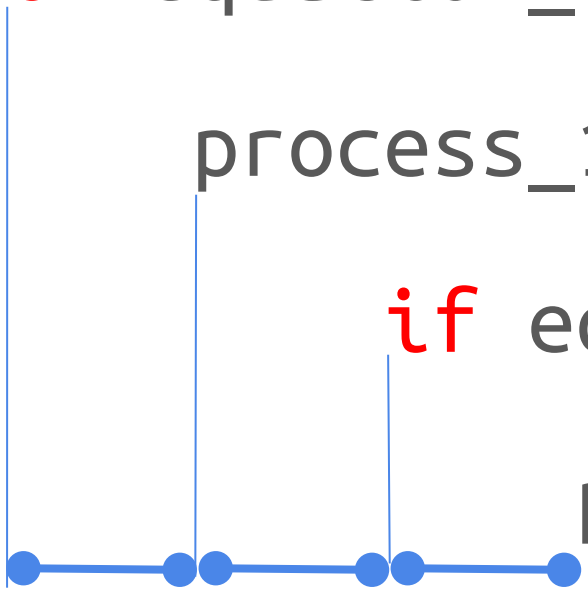
Pythonの基本

インデント　Pythonでは構文的な意味を持つ

```
if equation_A:
    process_1
        if equation_B:
            process_2
else:
    process_3
```

インデント　Pythonでは構文的な意味を持つ

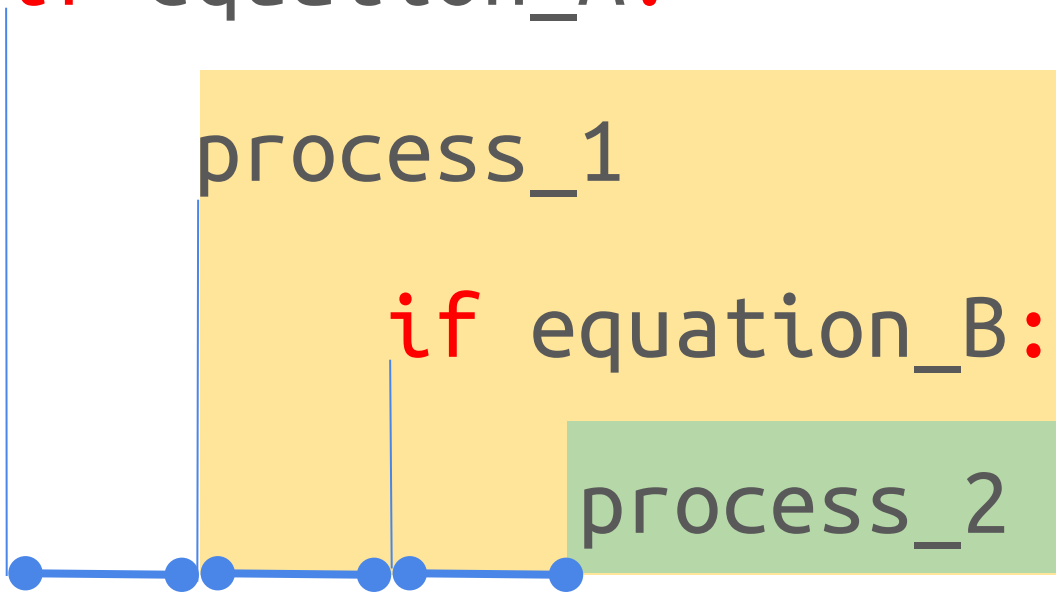
```
if equation_A:
    process_1
        if equation_B:
            process_2
else:
    process_3
```



＝インデント
（通常半角スペース4つ）

インデント　Pythonでは構文的な意味を持つ

```
if equation_A:
    process_1
        if equation_B:
            process_2
else:
    process_3
```



ブロック
制御構文からの一連の
処理の範囲

処理の範囲をインデントで判別
インデントがずれると想定外の動作 or エラー

```
IndentationError: expected an indented block
```

1) コメントと文字列

```
# から始まる文はコメント行
# コードとして認識されない
x = 2   # 文中にコメントも可
```

1) コメントと文字列

```
'1行の文字列'

"シングル or ダブルクォーテーション1つ"

"Let's study Python"
'Let\'s study Python'
```

1) コメントと文字列

'''

複数行の文字列
シングル or ダブルクォーテーション3つ
'''

"""

1行の文字列でもOK
"""

2) 数値、文字列と四則演算

数値と文字列

```
1000          # 整数
1000.0        # 浮動小数点数
10e-2         # 0.01; 浮動小数点数
"AMeDAS"      # 文字列
'メッシュ'    # 文字列
```

2) 数値、文字列と四則演算

四則演算の演算子

+	加算
-	減算
*	乗算
/	除算
//	除算（切り捨て）
%	剰余
**	指数

3) 四則演算と代入文

```
1 + 2    # 加算; 3
2 - 1    # 減算; 1
2 * 2    # 乗算; 4
6 / 3    # 除算; 2
7 // 3   # 除算(切り捨て); 2
7 % 3    # 除算(余り); 1
3 ** 2   # 指数; 9
```

3) 四則演算と代入文

```
'メッシュ' + '農業' + '気象'
# 'メッシュ農業気象'

'農業' * 3
# '農業農業農業'
```

3) 四則演算と代入文

```
lat = 36.0270
lon = 140.1104
site_name = 'noukanken'

site_name * 2   # 'noukankennoukanken'
lat + 5         # 41.027
site_name + 5   # error
site_name + '5'
```

3) 四則演算と代入文

```
a = 100


a += 10   # a = a + 10 と同じ
a -= 10   # a = a - 10 と同じ
a *= 10   # a = a * 10 と同じ
a /= 10   # a = a / 10 と同じ
```

4) 比較演算子

`x < y`	xはyより小さい
`x <= y`	xはyより小さい、もしくはxとyは等しい
`x > y`	xはyより大きい
`x >= y`	xはyより大きい、もしくはxとyは等しい
`x == y`	xとyは等しい(等価である)
`x != y`	xとyは等しくない（等価でない）
`x is y`	xとyは同じオブジェクトである
`x is not y`	xとyは同じオブジェクトではない
`x in y`	xはyに含まれる
`x not in y`	xはyに含まれない

4) 比較演算子

```
# 代入
a = [1, 2, 3]
# 値が同じか否か
a == [1, 2, 3]  # True
# 同じオブジェクトか否か
a is [1, 2, 3]   # False
```

4) 比較演算子

```
# 代入

0 < a < 50
0 < b <= 200
```

5) リスト 角括弧で囲む

```
[1, 2, 3]

['apple', 'orange', 'banana']

[1, 2.0, 'banana', values]

[]

# 数字始まりの変数は不可
```

5) リスト 角括弧で囲む

```
fruits = ['apple', 'orange', 'banana']

values = [1, 2, 3]

fruits + values

# ['apple', 'orange', 'banana', 1, 2, 3]
```

5) リスト 角括弧で囲む

```
fruits = ['apple', 'orange', 'banana']

fruits.append('mango')

# ['apple', 'orange', 'banana', 'mango']

fruits * 2

# ['apple', 'orange', 'banana', 'mango',
  'apple', 'orange', 'banana', 'mango']
```

5) リスト 要素の抽出

```
# Pythonでは要素をゼロから数える

                  0         1          2
fruits = ['apple', 'orange', 'banana']

fruits[0]  # ’apple’
```

5) リスト 要素の抽出

```
                0         1         2
fruits = ['apple', 'orange', 'banana']
         0        1         2         3
                (-2)      (-1)      (0)

fruits[0:2]   # ['apple', 'orange']

fruits[0:-1]  # ['apple', 'orange']
```

5) リスト 要素の抽出

```
          0    1          2                3
mixed =[1, 2.0, 'banana', [1, 2, 3]]
                                  0  1  2
mixed[3][2] # 3
```

5) リスト 要素の抽出

```
fruits = ['apple', 'orange', 'banana']

'orange' in fruits  # True


values = [1, 2, 3]

5 in values  # False
```

6) タプル 更新不能

```
(10, 20, 30)

('spinach', 'carrot', 'onion')

('spinach', 100, 5.0)

()

values.append(40) # error; 値の追加は不可
```

7) 辞書

```
{'apple':100, 'banana':50}
```

キー　値

```
{}
```

7) 辞書

```
fruits_price = {'apple':100, 'banana':50}


fruits_price['apple']  # 100

fruits_price['avocado'] = 140

fruits_price['apple'] = 200  # 上書き
```

9) 制御構文 for文による繰り返し

```
fruits = ['apple', 'banana', 'orange']


for fruit in fruits:
    # 変数fruitに格納された値を出力
    print(fruit)
```

9) 制御構文 range()関数との組み合わせ

```
# ゼロからn-1までの数列のイテレータ
range(n) ≒ [0, 1, 2, …, n-1]

# nからm-1未満、差分dの数列のイテレータ
range(n, m, d) ≒ [n, n+d, n+2d,..., n+xd]

# イテレータ = オブジェクトを列挙するための
               インターフェース
```

9) 制御構文 range()関数との組み合わせ

```
# range(3) ≒ [0, 1, 2]
for i in range(3):
    print(i)  # 0, 1, 2


# range(1, 10, 2) ≒ [2, 4, 6, 8]
for i in range(1, 10, 2):
    print(i)  # 1, 3, 5, 7, 9
```

9) 制御構文 リスト内包

```
cities = ['Tokyo', 'New York', 'Paris']

[len(city) for city in cities]

# len()は要素数を返す組み込み関数
# [5, 8, 5]
```

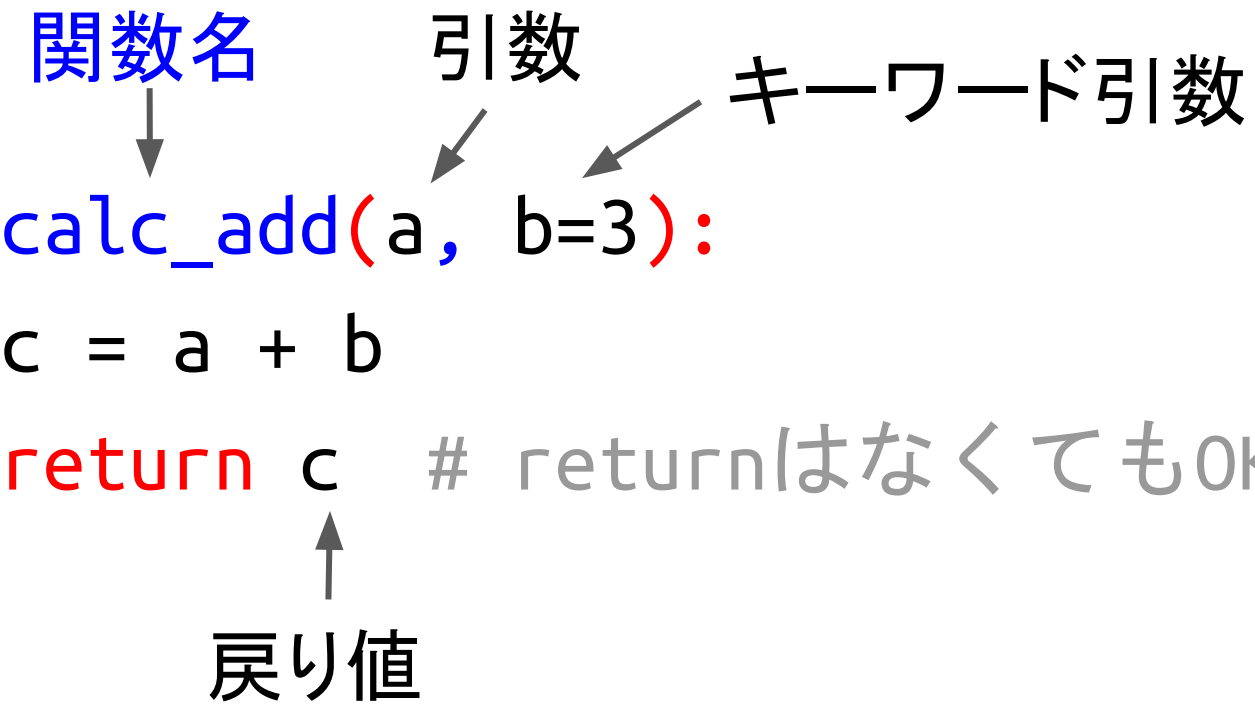
9) 制御構文 for文で書いた場合

```
cities = ['Tokyo', 'New York', 'Paris']


city_len = []
for city in cities:
    city_len.append(len(city))

city_len
```

10) 関数




```
calc_add(10, 20)
# 30
```

10) 関数 関数の呼び出し

```
"""
def calc_add(a, b=3):
    c = a + b
    return c
"""
calc_add(10, 20)  # 30
calc_add(10)      # 13
return_value = calc_add(10, 20)
return_value # 30
```

11) ライブラリのインポート

関数

モジュール　関数等のまとまり（拡張子pyのファイル）

パッケージ　モジュールのまとまり

ライブラリ　`import文`で読み込めるモジュール・パッケージ

11) ライブラリのインポート

```
# ライブラリのインポート
import numpy
```

ライブラリ・モジュール名

```
# 略称を指定してインポート
import numpy as np
```

略称

11) ライブラリのインポート

```
# ライブラリ内の関数を明示的にインポート
from math import floor
# import以下に複数の関数をカンマ区切りで列記可能
from math import floor, ceil
# 略称を指定してライブラリ内の関数を明示的にインポート
from math import floor as fr
# 略称を指定してライブラリ内のモジュールを明示的にインポート
from matplotlib.pyplot as plt
```

インポートしたライブラリの利用

```
numpy.zeros((2, 3))
np.zeros((2, 3))
floor(3.6)
ceil(3.6)
fr(3.6)
plt.figure()
```

2. NumPy

1) NumPyのインポート

```
# NumPy = Numerical Python の略語

# 数値計算を効率的に行うためのライブラリ

# 多次元配列（行列など）の操作が容易に

import numpy as np
```

2. NumPy

2) ndarray

```
# リスト
list_data = [1, 2, 3]

# リストをndarrayに変換
ndarray_data = np.array(list_data)
# ndarray_data = np.array([1, 2, 3])
```

2. NumPy

3) ndarrayの計算

```
# list_data = [1, 2, 3]
# ndarray_data = np.array(list_data)


list_data + list_data
# [1, 2, 3, 1, 2, 3]


ndarray_data + ndarray_data
# ndarray [2, 4, 6]
```

2. NumPy

3) ndarrayの計算

```
# list_data = [1, 2, 3]
# ndarray_data = np.array(list_data)


ndarray_data - ndarray_data
# ndarray [0, 0, 0]
ndarray_data * ndarray_data
# ndarray [1, 4, 9]
ndarray_data / ndarray_data
# ndarray [1., 1., 1.]
```

2. NumPy

4) 多次元配列と要素の計算

```
array_2d = np.array([[1, 2, 3],
                     [10, 20, 30],
                     [100, 200, 300]])
```


```
array_2d[行, 列]
```

2. NumPy

4) 多次元配列と要素の計算

```
array_2d[0, :]
array_2d[0]
```

	0列	1列	2列
0行	1	2	3
1行	10	20	30
2行	100	200	300

2. NumPy

4) 多次元配列と要素の計算

```
Array_2d[:, 0]
```

	0列	1列	2列
0行	1	2	3
1行	10	20	30
2行	100	200	300

2. NumPy

4) 多次元配列と要素の計算

```
Array_2d[1, 2]
```

	0列	1列	2列
0行	1	2	3
1行	10	20	30
2行	100	200	300