応用計量分析２（第3回）

担当教員: 梶野 洸（かじの ひろし）

本日の内容

Python に慣れようの回

- Python とは？
- 環境構築
- 基本的な文法
- 数値計算

Python とは？

- プログラミング言語の一つ
- 近年よく用いられる
    - 文法がわりと簡単
    - データ解析に関するライブラリが揃っている
    - 特にディープラーニングはほぼ Python

Python のライブラリ (1/2)

- `numpy`, `scipy`
  - 科学計算ライブラリ
  - 行列計算、統計処理など
  - python ブームの火付け役？
- `pandas`
  - 時系列解析ライブラリ
  - R みたいなやつ

Python のライブラリ (2/2)

- `keras`, `pytorch`, `chainer`
  - ディープラーニング用ライブラリ
  - ネットワークを組み替えるのが容易
  - ディープラーニングやるならこれらを使う
- `matplotlib`, `bokeh`, `plotly`
  - グラフを描くライブラリ

環境構築

どのようにコードを書いて実行するか

- `jupyter notebook` の利用を想定
    - コードの実行が簡単
    - 結果が見やすい
    - 起動の仕方は第一回の宿題参照

演習

Jupyter notebook を起動してください

基本的な文法

Hello, world!

In [2]:
```
print("Hello, world!")
```

```
Hello, world!
```

基本的な文法

困ったら print

In [3]:
```
print(1)
```

```
1
```

In [4]:
```
print(1 + 1)
```

```
2
```

In [5]:
```
x = 1
print(x)
```

```
1
```

In [6]:
```
x = 1
x # jupyter notebook では最後の変数の中身が勝手に表示される
```

Out[6]:
```
1
```

基本的な文法

四則演算

In [7]:
```
1 + 1 # 整数同士の足し算
```

Out[7]:
```
2
```

In [8]:
```
1.0 + 1.0 # 実数同士の足し算
```

Out[8]:
```
2.0
```

In [9]:
```
3 - 2 # 整数同士の引き算
```

Out[9]:
```
1
```

In [10]:
```
3 * 2 # 整数同士の掛け算
```

Out[10]:
```
6
```

In [11]:
```
5 / 3 # 割り算（Python3の場合、スラッシュで普通の割り算）
```

Out[11]:
```
1.6666666666666667
```

In [12]:
```
5.0 / 3.0 # 実数同士の割り算
```

Out[12]:
```
1.6666666666666667
```

In [13]:
```
5 // 3 # 整数同士の割り算で商を知りたい場合
```

Out[13]:
```
1
```

In [14]:
```
10 % 3 # 整数同士の割り算で余りを知りたい場合
```

Out[14]:
```
1
```

In [15]:
```
100 + 10 / 5 # 割り算は足し算より優先される
```

Out[15]:
```
102.0
```

In [16]:
```
(100 + 10) / 5 # カッコを使うと優先される演算を決められる
```

Out[16]:
```
22.0
```

In [17]:
```
2 ** 10 # 累乗は **
```

Out[17]:
```
1024
```

In [18]:
```
'Hello, world!' # ダブルコーテーション、シングルコーテーションで括ると文字列になる
```

Out[18]:
```
'Hello, world!'
```

In [19]:
```
'112313' # 整数(int)ではなく文字列(str)になる
```

Out[19]:
```
'112313'
```

In [20]:
```
'112313' + 5 # str と int の足し算はできない
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-292b60963b18> in <module>
----> 1 '112313' + 5 # str と int の足し算はできない

TypeError: must be str, not int
```

In [21]:
```
'112313' + '112313' # str 同士の足し算は、文字列の連結になる
```

Out[21]:
```
'112313112313'
```

演習

1. 税抜価格1980円の商品の税込価格は？（消費税8%）
2. 2^{2^2} を計算せよ
3. 好きな $x \in \mathbb{R}_+$ で $(1+x)^{1/x}$ を計算せよ

- ちなみに $e = \lim_{x \to 0}(1+x)^{1/x}$
  $= 2.718281828459045...$

演習（答え）

In [22]:
```
1980 * 1.08
```

Out[22]:
```
2138.4
```

In [23]:
```
2**(2**2)
```

Out[23]:
```
16
```

In [24]:
```
(1 + 0.000000001) ** (1 / 0.000000001)
```

Out[24]:
```
2.71828205201156
```

In [25]:
```
(1 + 0.0000000000001) ** (1 / 0.0000000000001)
```

Out[25]:
```
2.716110034086901
```

In [26]:
```
(1 + 0.000000000000001) ** (1 / 0.000000000000001)
```

Out[26]:
```
3.0350352065492614
```

In [27]:
```
(1 + 0.0000000000000001) ** (1 / 0.0000000000000001)
```

Out[27]:
```
1.0
```

演習（解説）

- コンピュータでは実数を近似的にしか扱えない
  - 大きい数 ＋ 非常に小さい数 では、非常に小さい数が無視される（丸め誤差）
  - 上の例では $1 + 0.0000000000000001 = 1$
- 計算方法の工夫が必要
  - x を小さくした極限で変な値になるのは困る

変数

数値、文字列、リストなどを記憶しておける

In [28]:
```
x = 1 # x に 1 を代入する
print(x)
```

```
1
```

In [29]:
```
x = 1
x = 2
print(x)
```

```
2
```

In [30]:
```
x = 1
y = 4
print(x * y)
```

```
4
```

In [31]:
```
x = 0
x = x + 1
print(x)
```

```
1
```

リスト

複数のオブジェクトをひとまとめにする

In [32]:
```
[1, 2, 6, 4, 8] # 大カッコでリストを宣言する
```

Out[32]:
```
[1, 2, 6, 4, 8]
```

In [33]:
```
x = ['yay!', 1, 3.0, [4, 0.1, 'yay!']] # リストの中身は、数値・文字列・リストなど
```

In [34]:
```
#x[3] # 0番目の要素を取り出す
x
```

Out[34]:
```
['yay!', 1, 3.0, [4, 0.1, 'yay!']]
```

In [35]:
```
#x[0 : 3] # 0番目から2番目までの要素を取り出す
x[0] = 0
x
```

Out[35]:
```
[0, 1, 3.0, [4, 0.1, 'yay!']]
```

タプル

中身が変更できないリストみたいなもの

In [36]:
```
x = (1,2,3) # ふつうの括弧でくくるとタプル
```

In [37]:
```
x[0] # リストみたいに中の要素にアクセスできる
```

Out[37]:
```
1
```

In [38]:
```
x[0] = 100 # 一度作ったら変更できない
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-38-802cdab161c2> in <module>
----> 1 x[0] = 100 # 一度作ったら変更できない

TypeError: 'tuple' object does not support item assignment
```

辞書

数字以外のもので値を取り出せる

In [39]:
```
x = {'W': 1, 'b': -0.1} # 中括弧で囲む& key: value と書くと辞書ができる
```

In [40]:
```
type(x)
```

Out[40]:
```
dict
```

In [41]:
```
x['W'] # キーを使ってそれに対応する値を取り出す。キーは文字列でもよい。
```

Out[41]:
```
1
```

In [42]:
```
x[(1, 2)] = 3 # このように新しい要素を追加できる。タプルをキーにすることもできる
x
```

Out[42]:
```
{'W': 1, 'b': -0.1, (1, 2): 3}
```

In [43]:
```
x[[1,2]] = 3 # リストはキーにできない
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-43-8777a0ab5a51> in <module>
----> 1 x[[1,2]] = 3 # リストはキーにできない

TypeError: unhashable type: 'list'
```

ここまでのまとめ

- 数値、文字列、リスト、タプル、辞書など基本的なオブジェクトを触った
    - 複数の値を持っておきたいとき、基本はリストか辞書
    - 整数以外のキーで要素を取り出したい場合は辞書
- 数値計算をすこしやった
- 大体のものは変数に入れておける

ここから

制御構文に慣れ親しむ

- if 文
- for 文
- while 文

条件・if文

ある条件が満たされたときだけ実行される

In [44]:
```
x = 1
print(x == 1) # x が 1 と等しいとき True
print(x != 1) # x が 1 ではないとき True
print(x != 0) # x が 0 ではないとき True
```

```
True
False
True
```

In [45]:
```
x = [1,1]
if x[0] == 0: # if (条件): が if 文
    print('yay!') # Python では必ず <tab> を打ってインデントを下げる！
    if x[1] == 1: # if 文の中に if 文を入れられる
        print('yayyay')
print('hello') # インデントが下がっていないので、ここからは if 文の外
```

```
hello
```

In [46]:
```
x = 1
if x == 0: # if (条件): が if 文
print('yay!') # かならず <tab> を打ってインデントを下げる！
```

```
  File "<ipython-input-46-e07010cf2b42>", line 3
    print('yay!') # かならず <tab> を打ってインデントを下げる！
        ^
IndentationError: expected an indented block
```

In [47]:
```
x = 1
if x == 0:
    print('x=0')
else: # if 文の条件がFalseの場合に実行される
    print('x!=0')
```

```
x!=0
```

In [48]:
```
x = [1,2]
if x[0] == 1:
    if x[1] == 0:
        print('x=0')
    elif x[1] == 1:
        print('x==1')
    else:
        print('x!=0 and x!=1')
```

```
x!=0 and x!=1
```

for 文

In [49]:
```
for i in range(5): # range(5) だと 0,1,...,4
    print(i)         # for 文内はインデントを下げる
```

```
0
1
2
3
4
```

In [50]:
```
for i in range(5): # range(5) だと 0,1,...,4
print(i)
```

```
  File "<ipython-input-50-0e78f8284010>", line 2
    print(i)
        ^
IndentationError: expected an indented block
```

In [51]:
```
for i in range(1,5): # range(1,5) だと 1,2,...,4
    print(i)
```

```
1
2
3
4
```

In [52]:
```
for i in range(0, 10, 3): # range(0, 10, 3) だと 3 ごとに
    print(i)
```

```
0
3
6
9
```

In [53]:
```
print(range(0,10) == [0,1,2,3,4,5,6,7,8,9]) # range はリストっぽいけどリストじゃない
print(list(range(0,10)) == [0,1,2,3,4,5,6,7,8,9]) # リストに変換することもできる
```

```
False
True
```

while 文

In [54]:
```
i = 0
while i < 10: # i<10 である限りインデント以下の操作を繰り返す
    i = i + 1
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

演習

- 0から99までの偶数を表示せよ
- 0から99までの3の倍数を表示せよ
- Fizz Buzz 問題(99まで)
    - 1,2,...から順に数を表示したい
    - 3の倍数のときは数字の代わりに 'Fizz' と表示
    - 5の倍数のときは数字の代わりに 'Buzz' と表示
    - 15の倍数のときは数字の代わりに 'FizzBuzz' と表示
    - 例: 1,2,Fizz,4,Buzz,...,13,14,FizzBuzz,16,...
    - ヒント: `range(1,100)` は1から99まで

1.0から99までの偶数を表示せよ

In [55]:

```
for i in range(100):
    if i % 2 == 0:
        print(i)
```

```
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
```

In [56]:
```
for i in range(0, 100, 2):
    print(i)
```

```
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
70
```

1. 0から99までの3の倍数を表示せよ

In [57]:

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

```
0
3
6
9
12
15
18
21
24
27
30
33
36
39
42
45
48
51
54
57
60
63
66
69
72
75
78
81
```

1. FizzBuzz問題

In [58]:

```
for i in range(1,100):
    if i % 3 == 0 and i % 5 != 0:
        print('Fizz')
    elif i % 3 != 0 and i % 5 == 0:
        print('Buzz')
    elif i % 3 == 0 and i % 5 == 0:
        print('FizzBuzz')
    else:
        print(i)
```

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
Fizz
22
23
Fizz
Buzz
26
Fizz
28
29
```

関数

何度も使う手続きを関数として定義しておける

関数: 入出力関係を表したもの

- 入力 = 引数
- 出力 = 返り値

と呼ぶ

In [59]:
```
# 引数 = x, 返り値 = x + 1
def increment_one(x): # def 関数名(引数):
    return x + 1
```

In [60]:
```
increment_one(x=-1)
```

Out[60]:
```
0
```

In [61]:
```
# 引数 = x, y, 返り値 = z = x + y
def add(x, y):
    z = x + y
    return z
```

In [62]:
```
print(add(5, 10))
print(add(add(5, 10), 20))
```

```
15
35
```

In [63]:
```
def add_y_ntimes(x, y, n):
    if n == 1:
        return x + y
    return add_y_ntimes(x + y, y, n - 1) # 関数の中で自分を呼んでも良い（再帰的呼び出し）
```

In [64]:
```
print(add_y_ntimes(10, 1, 1))
print(add_y_ntimes(10, 1, 5))
```

```
11
15
```

In [65]:
```
# 何も返さなくても良い
def fizz_buzz(max_iter):
    for i in range(1, max_iter+1):
        if i % 3 == 0 and i % 5 != 0:
            print('Fizz')
        elif i % 3 != 0 and i % 5 == 0:
            print('Buzz')
        elif i % 3 == 0 and i % 5 == 0:
            print('FizzBuzz')
        else:
            print(i)
```

In [66]:
```
fizz_buzz(10)
```

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
```

演習

数値が入ったリストを入力したときに、小さい順に並べ直したリストを出力する関数を書け

ヒント

クイックソート

In [67]:
```
def quicksort(x):
    if len(x) < 2:
        return x
    else:
        left_list = []
        right_list = []
        num_pivot = 0
        pivot = x[0]

        for each_element in x:
            if each_element < pivot:
                left_list.append(each_element)
            elif each_element > pivot:
                right_list.append(each_element)
            else:
                num_pivot = num_pivot + 1
        return quicksort(left_list) + num_pivot * [pivot] + quicksort(right_list)
```

In [68]:
```
quicksort([-1, 2, 1, 1, 1,3])
```

Out[68]:
```
[-1, 1, 1, 1, 2, 3]
```

In [69]:
```
quicksort([1,2,3,4,5]) == [1,2,3,4,5]
```

Out[69]:
```
True
```

まとめ

- 基本的なデータ型を触った
    - 単体: 数値、文字列
    - 複数の値を格納するもの: リスト、辞書、タプル
- 数値計算をちょっとやってみた
    - 丸め誤差には注意

- 制御構文を触った
    - for文、if文、while文
    - FizzBuzz
    - 一段インデントを下げる！
- 関数を定義した
    - よく使う作業をひとかたまりにする
    - 一段インデントを下げる！

質問