# 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`, `tensorflow`
  - 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

- `repl.it` の利用を想定
    - コードの実行が簡単
    - 環境セットアップがいらない
    - 理解度確認テストとの兼ね合い

# 演習

1. https://repl.it (https://repl.it) を開く
2. GitHub アカウントでログインする（ログインしている場合はスキップ）
3. `create a repl` と書いてあるところの下の `Python` という箇所をクリック

# 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
```

# 基本的な文法

四則演算

色々演算が定義されている

In [6]:
```
1 + 1 # 整数同士の足し算
```

Out[6]:
```
2
```

In [7]:
```
1.0 + 1.0 # 実数同士の足し算
```

Out[7]:
```
2.0
```

In [8]:
```
3 - 2 # 整数同士の引き算
```

Out[8]:
```
1
```

In [9]:
```
3 * 2 # 整数同士の掛け算
```

Out[9]:
```
6
```

In [10]:
```
5 / 3 # 割り算（Python3の場合、スラッシュで普通の割り算）
```

Out[10]:
```
1.6666666666666667
```

In [11]:
```
5.0 / 3.0 # 実数同士の割り算
```

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

In [12]:
```
5 // 3 # 整数同士の割り算で商を知りたい場合
```

Out[12]:
```
1
```

In [13]:
```
10 % 3 # 整数同士の割り算で余りを知りたい場合
```

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

In [14]:
```
2 ** 10 # 累乗は **
```

Out[14]:
```
1024
```

計算順序は数学と同様

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
```

文字列と数字は異なるものとして扱われる

In [17]:
```
'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[17]:
```
'Hello, world!'
```

In [18]:
```
'112313' # 整数(int)ではなく文字列(str)になる
```

Out[18]:
```
'112313'
```

In [19]:
```
'112313' + 5 # str と int の足し算はできない
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-292b60963b18> in <module>
----> 1 '112313' + 5 # str と int の足し算はできない

TypeError: can only concatenate str (not "int") to str
```

In [20]:
```
'112313' + '112313' # str 同士の足し算は、文字列の連結になる
```

Out[20]:
```
'112313112313'
```

# 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。 y = f(x) という数学の関数と大体同じ（xが引数、yが返り値）。

# 関数の作りかた

In [21]:
```
def increment_one(x0): # def 関数名(引数):
    return x0 + 1
increment_one(x0=0) # 引数の x という変数に 0 を代入して実行
```

Out[21]:
```
1
```

In [22]:
```
print(x0) # 上の関数の引数で x0 という変数が定義されていたが、それは関数の外では使えない
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-3ef74515dee8> in <module>
----> 1 print(x0) # 上の関数の引数で x0 という変数が定義されていたが、それは関数の外では使えない

NameError: name 'x0' is not defined
```

- 関数名: `increment_one`（好きな名前を決める）
- 引数: `x`
    - 引数は **関数内のみ** で使える変数となる
    - 変数には最悪何が入ってくるかはわからないが、関数設計時には何を入力したらいいかの想定はする
- 返り値: `x + 1`

# 演習3.0

main.py に以下の関数を実装してください。URLはLETUS参照。

1. identity
    - 関数名: identity
    - 引数: 1つ
    - 返り値: 引数をそのまま返す
2. double
    - 関数名: double
    - 引数: 1つ
    - 返り値: 引数を2倍したものを返す
3. addition
    - 関数名: addition
    - 引数: 2つ
    - 返り値: 2つの引数を足したものを返す

# 演習3.1

- 課題1: 税抜き価格を入力として、税込価格（消費税8%、整数でなくてもOK）を出力する関数 `assignment1` を完成させよ。
- 課題2: `base` と `exponent` を入力として、 `base` の `exponent` 乗を出力する関数 `assignment2` を完成させよ。
- 課題3: `x` を入力として、 (1+x) の 1/x 乗を出力する関数 `assignment3` を完成させよ。
    - ちなみに e = $\lim_{x \to 0}(1 + x)^{1/x}$ = 2.718281828459045...

In [29]:
```
(1 + 0.000000001) ** (1 / 0.000000001)
```

Out[29]:
```
2.71828205201156
```

In [30]:
```
(1 + 0.0000000000001) ** (1 / 0.0000000000001)
```

Out[30]:
```
2.716110034086901
```

In [31]:
```
(1 + 0.000000000000001) ** (1 / 0.000000000000001)
```

Out[31]:
```
3.0350352065492614
```

In [32]:
```
(1 + 0.0000000000000001) ** (1 / 0.0000000000000001)
```

Out[32]:
```
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 [33]:
```
x = 1 # x に 1 を代入する
print(x)
```

```
1
```

In [34]:
```
x = 1
x = 2
print(x)
```

```
2
```

In [35]:
```
x = 1
y = 4
print(x * y)
```

```
4
```

In [36]:
```
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 [37]:
```
[1, 2, 6, 4, 8] # 大カッコでリストを宣言する
```

Out[37]:
```
[1, 2, 6, 4, 8]
```

In [38]:
```
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 [39]:
```
#x[3] # 0番目の要素を取り出す
print(x[0])
```

```
yay!
```

In [40]:
```
x[0 : 3] # 0番目から2番目までの要素を取り出す
```

Out[40]:
```
['yay!', 1, 3.0]
```

In [6]:
```
x = [] # 空リストを作る
x.append(3) # 3 を末尾に追加する
print(x)
x.append(5)
print(x)
print(x.append(6)) # append というメソッド自体は何も値を返さない
print(x)
```

```
[3]
[3, 5]
None
[3, 5, 6]
```

# タプル

中身が変更できないリストみたいなもの

In [41]:
```
x = (1,2,3) # ふつうの括弧でくくるとタプル
```

In [42]:
```
x[0] # リストみたいに中の要素にアクセスできる
```

Out[42]:
```
1
```

In [43]:
```
x[0] = 100 # 一度作ったら変更できない
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-43-802cdab161c2> 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 [44]:
```
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 [45]:
```
type(x)
```

Out[45]:
```
dict
```

In [46]:
```
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[46]:
```
1
```

In [47]:
```
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[47]:
```
{'W': 1, 'b': -0.1, (1, 2): 3}
```

In [48]:
```
x[[1,2]] = 3 # リストはキーにできない
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-48-8777a0ab5a51> in <module>
----> 1 x[[1,2]] = 3 # リストはキーにできない

TypeError: unhashable type: 'list'
```

# 演習3.1.1

1. リスト input_list と非負の整数値 idx を入力とし、input_list の idx 番目の要素を出力する関数 assignment1 を完成させよ。ただし idx は 0 以上 len(input_list)-1 以下であることを前提とし、input_list の一番はじめの要素を「0番目の要素」とする。
2. 辞書 input_dict と任意の key を入力とし、input_list の idx 番目の要素を出力する関数 assignment2 を完成させよ。ただし key は input_dict のキーの集合に含まれているとする。
3. リスト input_list と任意のオブジェクト obj を入力とし、input_list の末尾に obj を追加したリストを出力する関数 assignment3 を完成させよ。

# 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 [49]:
```
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 [50]:
```
x = 0
# if (True/Falseを返す条件):
#     True ならば実行するコード（Falseの時は実行されない）
if x == 0: # もし x が 0 と等しいならば、以下を実行する
    print('yay')
    print('hello world')
print('yay!!!!!!!!!')
```

```
yay
hello world
yay!!!!!!!!!
```

In [51]:
```
x = [0,0]
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
```

```
yay!
```

In [52]:
```
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-52-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 [53]:
```
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 [54]:
```
x = 1
if x == 0:
    print('x=0')
elif x == 1:
    print('x=1')
else:
    print('x!=0 and x!=1')
```

```
x=1
```

# for 文

In [55]:
```
for i in [0, 1, 2, 3, 4]: # for <変数名> in <なんかリスト的なもの>:
    print(i)
```

```
0
1
2
3
4
```

In [56]:
```
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 [57]:
```
for i in range(5): # range(5) だと 0,1,...,4
print(i)
```

```
  File "<ipython-input-57-0e78f8284010>", line 2
    print(i)
    ^
IndentationError: expected an indented block
```

In [58]:
```
for i in range(1, 5): # range(1,5) だと 1,2,...,4
    print(i)
```

```
1
```

In [59]:
```
for i in range(0, 10, 3): # range(0, 10, 3) だと 3 ごとに
    print(i)
```

```
0
3
6
9
```

In [60]:
```
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 [61]:
```
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3.2

1. `max_int`（0以上の整数）を入力とし、0から`max_int`まで（`max_int`を含む）の偶数が昇順に並んだリストを出力する関数`multiple_two`を完成させよ
2. `max_int`（0以上の整数）を入力とし、0から`max_int`まで（`max_int`を含む）の3の倍数が昇順に並んだリストを出力する関数`multiple_three`を完成させよ
3. `max_int`（0以上の整数）を入力とし、以下に定義するFizzBuzzゲームの結果を`max_int`回まで実行した結果からなるリストを出力する関数`fizzbuzz`を完成させよ

    - 1, 2, から順に数を言うゲームである
    - 3の倍数の時は数字の代わりに'Fizz'と言う
    - 5の倍数の時は数字の代わりに'Buzz'と言う
    - 15の倍数の時は数字の代わりに'FizzBuzz'と言う（上2つよりこのルールが優先される）
    - 例）`max_int=5`の時は`[1, 2, "Fizz", 4, "Buzz"]`というリストを返す

# 1. FizzBuzz問題

In [65]:
```
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
FizzBuzz
31
32
Fizz
34
Buzz
Fizz
37
38
Fizz
```

```
Buzz
41
Fizz
43
44
FizzBuzz
46
47
Fizz
49
Buzz
Fizz
52
53
Fizz
Buzz
56
Fizz
58
59
FizzBuzz
61
62
Fizz
64
Buzz
Fizz
67
68
Fizz
Buzz
71
Fizz
73
```

# その他の技法

In [66]:
```
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 [67]:
```
print(add_y_ntimes(10, 1, 1))
print(add_y_ntimes(10, 1, 5))
```

```
11
15
```

In [68]:
```
# 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 [69]:
```
fizz_buzz(13)
```

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

# 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 [70]:
```
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 [71]:
```
quicksort([-1, 2, 1, 1, 1,3])
```

Out[71]:
```
[-1, 1, 1, 1, 2, 3]
```

In [72]:
```
quicksort([1,2,3,4,5]) == [1,2,3,4,5]
```

Out[72]:
```
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