プログラミングII

黒瀬　浩

kurose@neptune.kanazawa-it.ac.jp

OH: 講義の前後，Eメール問合せ，月3限21-405

居室　67・121

第1回復習　辞書，辞書のメソッド

```
d = dict(a=1, b=2, c=3, d=4)        # 変数dとキーdは別物
```
(d = {"a":1, "b":2, "c":3, "d":4} と書いても上と同じ)

キー一覧	d.keys() ⇨*1	['a', 'b', 'c', 'd']
値一覧	d.values() ⇨*1	[1, 2, 3, 4]
ペア一覧	d.items() ⇨*1	[('a',1), ('b',2), ('c',3), ('d',4)]
要素数	len(d)	

1つづつ処理する書き方

```
for k,v in d.items():
    print("key=", k, "val=", v)
# items()メソッドで得られたタプルの1個目が変数kに，2個目が変数vに入る
```

並び替え(1番上の変数dを定義して以下を確認せよ)

キー昇順	sorted(d.items())
キー降順	sorted(d.items(), reverse=True)
値昇順	sorted(d.items(), key=lambda x:x[1])
値降順*2	sorted(d.items(), key=lambda x:x[1], reverse=True)

*1: 正確には，list(d.keys()) のようにリスト化が必要

*2: lambdaはその場のみで有効な関数を定義する(後述)

第2回復習 map, reduce, filter　　無名関数lambda

map　全ての要素に同じ演算を行う
reduce 要素を順に演算し集約する(functoolsパッケージ)
filter　条件に合うものを残す

上記は，第1引数に演算の無名関数を，第2引数に対象を指定する
いずれも，for文やlist()などを使わないと値が解らない(遅延評価)

```
map( lambda x: x*x, range(1,11) )    # 数列をそれぞれ2乗
import functools as ft # recude はimportが必要
ft.reduce( lambda x,y: x+y, range(1,11) ) # 数列の総和
filter( lambda x: x%2==0, range(1,11) ) # 条件による抽出
```

map,reduce,filterはリスト内包や関数でも実装可能

```
[x*2 for x in range(11)]                    # 2倍の数列
sum( range(11) )                            # 総和
[x for x in range(11) if x%2==0]            # 偶数のみ残す
```

math.sin()は，引数は数値だがnumpy.sin()はリストを渡せる map機能あり

第3回復習

イテレータ(iterator)

- iter()関数　で1つづつ値を返すオブジェクトを作れる
- next(イテレータオブジェクト)　で1つづつ値を取り出せる
- 終わりまでいくとStopIterationエラーが返る
- forはイテレータを取り出す動作を行う

ジェネレータ(generator)

- 関数で定義するが，returnでなくyieldで値を返す
- 関数は終了しないので関数内の変数は保持される
- 関数でやることがなくなる（またはreturn)でジェネレータは消滅

range()関数　整数列生成

- 引数1個　0から終了の1つ前まで
- 引数2個　開始から終了の1つ前まで
- 引数3個　開始から終了の1つ前まで指定間隔ごと

可変長引数

- 引数の数が異なっても受けられる
- 受け取る変数に*をつける（*の付いていないものより後に指定)
- タプルで渡される

第4回復習

処理継続できないエラーが出るとプログラム強制終了となる

```
try：
    試す処理
except:
    例外発生時の処理
```

で例外発生時に処理を継続できる

```
except 例外種類:
    発生した例外ごとの処理をかく
else:
    例外が発生しなかった場合の処理を書く
finally:
    例外が発生しても，しなくても実行する処理を書く
```

`raise` 例外種類（メッセージ） で例外を意図的に発生できる

モジュール（パッケージ，ライブラリ）

似たような機能をまとめてわかりやすくする
モジュールがないと，関数が増えてしまいわかりにくい
よく使われる関数は，特に何もしなくても使える
　　print(), input(), max()など

標準で提供されているが，指定しないと使えないもの
指定　　import パッケージ名
例　　`import math`
　　`# mathパッケージの関数sin()と定数piを使用`
　　`math.sin(math.pi) ⇨  1.2246467991473532e-16`

正弦関数や円周率は，多くのプログラムで使われるわけではないので
指定しないと使えないようになっている

import

```
import パッケージ名                          # パッケージを使えるようにする
# 以降 パッケージ名.関数名 や パッケージ名.変数 を使用できる

import パッケージ名1, パッケージ名2 # 複数のパッケージをimport
import パッケージ名 as 別名                  # 別名をつけられる

from パッケージ名 import 対象                # 対象を * にすると全部
# from では パッケージ名を指定しなくてよくなる
```

例

```
import math
math.pi  ⇨ 3.141592653589793
from math import *
pi ⇨ 3.141592653589793
math.pi is pi ⇨ True    # 同一であることが確認できた(==は同値)
```

fromを使うと短く書けるが，複数のパッケージで同じ名前で中身が違うものがあるので注意

pythonファイル名の注意

pythonファイル名にパッケージ名を使用してしまうと問題を起こす

例
math.py　というファイルを作る
そのディレクトリで別のファイルから

```
import math
```

を行うと，数学パッケージでなく，作成したmath.pyを読み込む
作成したmath.pyに文法エラーがなければ読み込みは成功するので
エラーは表示されない
math.sin()やmath.piを参照しても，見つからないというエラーになる
パッケージ検索パスの検索順は

```
import sys
sys.path
```

で確認できる
はじめに''があれば現在のディレクトから確認する

Python標準ライブラリ

追加インストールしなくても使える
たくさんあるので必要に応じて調べる
https://docs.python.org/ja/3/library/index.html
(このページの組み込み型はパッケージでなくクラスなのでimport不要)

パッケージ例

日付時間関係	datetime
乱数	random
数学	math
システムパラメータ	sys
OS関係	os

演習

1．整数nまでの一様乱数から出現頻度を数えよ

```
import random
n=10
for i in range(n): print( random.randint(1,n), end=' ' )
```

結果をキーにして出現数をカウントし表示するように変更せよ

一様乱数なので出現頻度は理想的には同じ

上記をさらに繰り返して（外側にループをつくり）出現頻度が同程度になるには何回くらい必要か確認せよ

2．nのべき乗の値を増やしていき総積処理の実行時間の増加傾向をみよ

　できたら出力型はtimedeltaなので，秒単位に変換せよ（時，分の桁に注意）

```
import datetime
n, m = 10**3, 1
begin = datetime.datetime.now()
for i in range(1, n+1):
      m *= i
print( datetime.datetime.now()-begin )
```

演習　コマンド引数の取得

総和を求めるプログラムで，上限が変わるごとにソースを編集するのは面倒．プログラム実行時に数を指定できるようにすれば，いちいちソースを見なくても良くなる（現実的に利用者は中身は見ていられない）

以下のファイルをつくれ

```
import sys
for i, j in enumerate( range( len(sys.argv) ) ):
    print(i, sys.argv[j] )
```

端末アプリケーションで作成したファイルを実行し出力を確認せよ

python ファイル名↵

python ファイル名 aaa↵

python ファイル名 aaa bbb↵

python ファイル名 aaa bbb ccc↵

引数が指定されなかったメッセージを出して終了するよう変更せよ

標準ライブラリ以外のものを使う

*1 https://pypi.org
*2 https://www.anaconda.com

python提供者はpython内部の機能拡張を主に行う
応用分野ごとに必要とする人々がパッケージを作成している
標準ライブラリ以外のものは，pypi(python package index)*1で提供される場合が多い．他にgithubやwebpageなどもある

anaconda*2は多くのパッケージ，開発ツールをまとめて提供している

pypiからパッケージを追加インストールする
最初に python対話型でimportしてエラーがなければ既に導入されている
import numpy　# numpyが入っているか確認

端末アプリケーションから
pip install –U パッケージ名
(学内では上記に　--proxy=wwwproxy.kanazawa-it.ac.jp:8080 もつける)
pipが古いというエラーがでたら，　pipでpipを更新する

anacondaのパッケージ管理

anacondaは主要な追加パッケージは既に含まれている
anacondaはいくつかのパッケージをまとめてからリリースされるので
pip(pypi)よりバージョンが低いことが多い

anacondaを使用している環境でpipコマンドで既に入っているパッケージを更新すると，バージョン不整合を起こすことがある
そのため，anaconda環境ではpipコマンドは使わない方が無難

端末アプリケーションより
conda install パッケージ名↵　　で追加インストールする
conda update –all↵　　で導入済みパッケージの一括更新をする

condaが古いというエラーがでたら
conda update conda↵　　でconda自体を更新する

パッケージ追加

anaconda利用者は既に入っているはずなので動作確認のみ実施

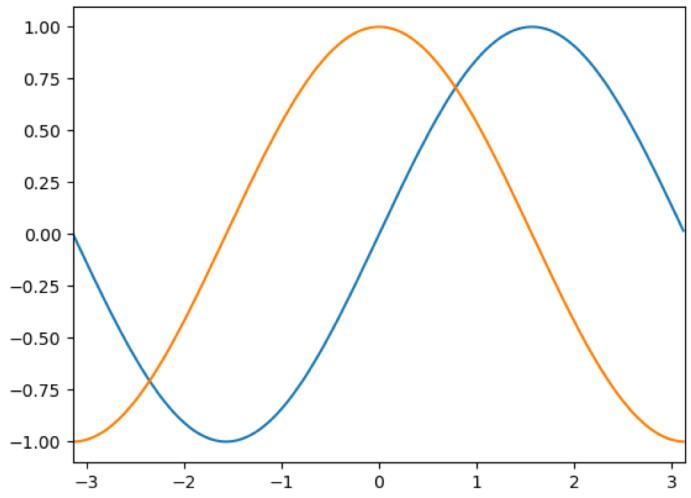
numpy, matplotlib, ipython の追加インストール
端末アプリケーションより

```
pip install -U --proxy=wwwproxy.kanazawa-it.ac.jp:8080 numpy
matplotlib ipython↵ （この2行は同じ行）
```

インストールできたら端末アプリケーションから

```
ipython↵
```

```
import matplotlib.pyplot as P, numpy as
x=N.arange(-N.pi, N.pi, 2*N.pi/360.0)
P.plot( x, N.sin(x) )
P.plot( x, N.cos(x) )
P.xlim( -N.pi, N.pi )
P.show()
```



ファイル入出力

プログラム内のデータはプログラムが終了すると無くなってしまう
プログラムが終了してもデータを保持するためにはファイルに書く
ファイルを使うと，プログラムを小さな単位に作ることができる

プログラムA ⇨　ファイル1 ⇨　プログラムB ⇨　ファイル2 ....
実際以下のことは既にやっている
テキストエディタ　⇨　pythonファイル　⇨　python ⇨　結果表示

ファイルを使うには作法がある
ファイルは他のアプリケーションでも使っているかもしれない
使用中に別のプログラムがデータを変更するかもしれない
使用中に消されるかもしれない

使うことを明示する　　open
終了することを明示する　　close

実際は処理をオペレーティングシステム(OS)に依頼して問題がおきないようにしている

open

open(ファイル名)
open(ファイル名, オプション)

ファイル名は文字列でファイルの場所と名前を指定する
'aaa.txt'　　実行した場所の
'c:¥Users¥tato¥prog2¥aaa.txt'　　完全進路名
'..¥..¥aaa.txt'　　相対進路名
オプション
mode=文字列　ファイルの読み書き指定
　r か w か a　　r:読み込み, w:上書き, a:追記
　の後にbを書くとバイナリ(非印字文字)も扱う指定(指定しなければt:テキスト)
　が指定できる 具体的には mode='r', mode='wb' など
encoding=文字列
　漢字コードの指定 encoding='sjis' など
　標準はutf-8だがコマンドプロンプトでは文字化けする
　コマンドプロンプト側でもchcpコマンドで表示コードを変えられる

open後の操作

```
a=open('aaa.txt')
type(a) ⇨ _io.TextIOWrapper # _ioパッケージのTextIOWrapper型
```

aは入出力を行うために必要となる
同じファイルでも別にopenすると別物となる

```
b=open('aaa.txt')
a is b ⇨ False
```

書き込み　　a.write("aaaaa")
読み込み　　data = a.read()
終了　　a.close()

write()はprint()と違い最後に改行をつけない（必要なら自分でつける）

```
data = open('aaa.txt').read()
```

とするとopenしてreadを1行で行える

withによるファイル利用区間明確化

```
with open('aaa.txt', mode='w') as f:
    for i in range(10):
        f.write( '%5d¥n'%(i) )  # 改行は自分で入れている
# ここで自動的にcloseされる
```

ファイル入出力はまとめてやったほうが効率が良いので上記は

```
with open('aaa.txt', mode='w') as f:
    s = ''
    for i in range(10):
        s += %5d¥n'%(i) )
    f.write( s )
```

のように文字列に追加していき，最後にwriteする方法がある

演習 プログラムリスト出力

適当なpythonソースファイルを作り，拡張子は.txt で保存

作成したファイルを読み込み，各行に行番号を　整数4桁頭0詰め　で
別のファイル（拡張子.txt)に出力するプログラムをつくれ

ヒント：　一括して読んだデータは改行コードで分割する　か
　　行毎に読む（何行あるか不明なのでwhileで脱出する方法を確認）

出力例（端末アプリケーションから　type ファイル名↵ で確認)

```
0001 def aaa(n):
0002       for i in range(n):
0003             print(i)
0004
0005 aaa( int( input("n? ") ) )
0006
```