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

レポート1　期限第3週開始時　紙で提出

第1回資料参照

選挙投票集計　または　目次生成　どちらかを選択
いずれも入力項目は 10以上にする(投票は10票以上，目次は10項目以上)
投票，目次生成とも同じものを10個入力しないこと（候補者，目次レベル）
pythonで書くこと

レポートタイトル(上部中央)　プログラミング2レポート1 選んだ方のタイトル
クラス名列，指名，提出日(上部右)
A4用紙 1枚(両面可，横長配置可）
ソースと実行結果のハードコピー(スナップショット）をワード等に貼る

他者にわかりやすいように書く
小さくて字が読めない場合や画像に余白が多い場合は減点する

ハードコピーは　altキーと printscreen(機種によってはprtsc)キー　で
クリップボードにコピーされるので，ワード等で貼り付ける
余計な部分を取り除くには画像を右クリックして　トリミング　を使う

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

遅延評価(lazy evaluation)

プログラミング言語には遅延評価に対応しているものがある

例えば

ものすごく大きな保管場所や処理時間を必要とする場合

```
for i in range(10**100):
    pass
```

0から$10^{100}-1$までの正数列を作成しないと処理できないので

必要になるまで処理を行わないようにすれば，必要とされたら値を返すことで，処理時間，メモリ資源を節約できる

対話型で range(10**100)を実行すると

```
>>> range(10**100)
range(0,
1000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000)
```

range型オブジェクトを返し，正数列を返さない

for で取り出した時に初めて正数列にする(1つずつ)

プログラム内では変数，定数，関数など全てオブジェクトとして管理される

iterator(反復子)

```
for i in range(10):
    print(i)
```

for文は繰り返し（反復）処理を実現する

順に取り出す機能を持つ

range()は指定した範囲の数列をつくる

自分でイテレータを作る

```
a=iter( [1,3,2,4] )        # リスト⇨イテレータ
next(a)    # aから1つ取り出す ⇨1
next(a)    # aから1つ取り出す ⇨3
```

もう取り出すものがなくなるとエラー(StopIteration)

for文はエラーがでたらブロックを抜ける処理となっている

エラーが出ても異常終了させない方法は例外処理で確認する

ジェネレータ (教科書P144参照)

関数はリターンすると関数内の変数は無くなってしまう
yieldは値は返すが関数は存続する（やることがなくなるまで）

```
def gen1to3():
    yield 1; # 最初のnext()でここまで動く
    yield 2; # 次のnext()でここまで動く
    yield 3; # その次のnext()でここまで動く
             # もうやることがないのでなくなる


a=gen1to3()  # ジェネレータ生成
type(a)    # ⇨ yeidを持った関数の型は　generator
next(a)    # ⇨ 1
next(a)    # ⇨ 2
next(a)    # ⇨ 3
next(a)    # ⇨ StopIterationエラー
```

関数内の変数がなくならないからできること

教科書 P146の例(少し変更)

```
def genOdd():
    i = 1
    while i<=30:
        yield i     # forで取り出すごとに値を返す
        i+=2

for v in genOdd():
    print(v, end=' ')
```

出力

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29
```

iが30を超えるとforで取り出せなくなる(StopIterationエラー)

for文はブロックを抜ける

自分でエラーを受け取る処理は次回に行う

returnで値を返すと変数iが無くなってしまうので呼び出し側で値を渡す必要があるが，generatorでは値を覚えているので呼び出し側は気にしなくて良い

yieldを使い range()関数と同様な処理を作れ

range()関数は，整数列を作る
1) 引数が1つなら第1引数から第2引数-1までの整数列
2) 引数が2つなら第1引数から第2引数-1までの整数列
3) 引数が3つなら第1引数から第2引数-1まで第3引数おきの整数列
4) 第3引数が負なら降順の整数列

当然だが，作成する関数内ではrange()は使えない
わからない人は，教科書のP144から147の例を先に行う

引数の数に応じて,range1(), range2(), range3()　として良い

できた人は，引数が1,2,3個でも対応できる関数range0()を作れ
ヒント：　def range0(*x): #可変長引数にする
　xはタプルとなる　len(x)で引数がいくつ指定されたかわかる
　x[0], x[1]などで個々の要素をアクセスする
　引数が1つのrange(n) は　range(0,n,1)と同じ
　引数が2つのrange(n,m)は　range(n,m,1)と同じ