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

複数のデータを格納する型の比較

型	変換関数	記法例	特徴など
文字列	`str()`	`'abc'`	変更不可
タプル	`tuple()`	`(1,2,3), tuple([1,2,3])`	変更不可
リスト	`list()`	`[1,2,3], list([1,2,3])`	
辞書	`dict()`	`{'a':1, 'b':2}, dict(a=1, b=2)`	キー重複不可
集合	`set()`	`{1,2,3}, set([1,2,3])`	値重複不可

長さを求める　　len()
値を参照する　　変数名[オフセットまたはキー]　　集合は不可
値を更新する　　変数名[オフセットまたはキー]=値　リストか辞書のみ可
演算子，メソッドは型により対応・動作が異なる

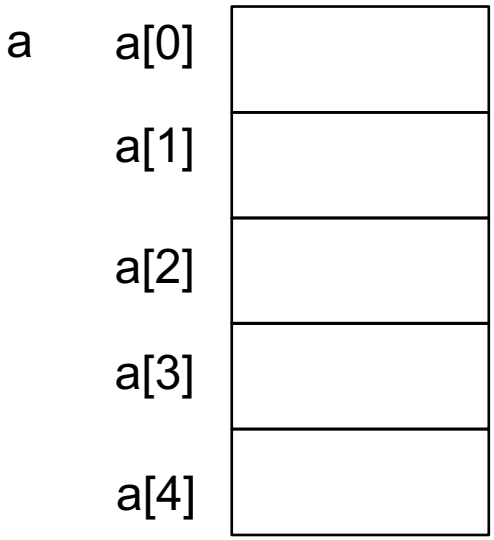
他にclassを使う方法もあるが説明は省略

ソースコードを読むとき誤解しないこと
(1,2,3) はタプル　変数名*1(1,2,3) は関数呼び出し
[1]はリスト　　変数名*1[1]は変数のオフセット1を参照

*1 正確には識別子(identifier)

参考：データの古典的格納方法　配列

プログラムで扱うデータをどのように配置するかは，性能・機能に依存する
C言語やJava言語の配列(Array)
int a[5]; // 整数型の値が格納できる配列を用意する



同じ大きさの器

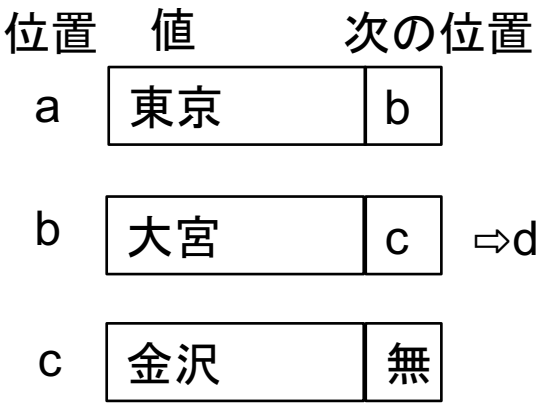
位置の計算
aの開始位置+何番目(オフセット）×1つの大きさ

作成時に大きさを決めておく必要がある
途中が不要となっても消せない
後で追加したくてもできない

全部の中から探す場合
格納順が不明なら
要素数nのとき平均検索時間はn/2

ランダウの記号　O(n/2)
Oはオーダー(Oder)の略
並べる順や検索方式によりO(n^2)，O(log n),
O(1)などがある
性能の指標となる

参考：途中が抜けたり，追加できるデータ格納方法(リスト)




```
a=['東京', '大宮', '金沢']
```

変数名はメモリの格納位置と対応している

大宮と金沢の間に長野を追加する
先の配列の例だと，大宮以降を最後から順にひとつづつずらして最後の空いたところに長野を入れる（あらかじめどの位追加するか分かっていなとずらせない）

リストでは，dに長野を作る．dの次の位置をc(金沢）にする．bの次の位置をdにする．


```
a.insert(2, '長野')
```

リストの要素は配列と異なり連続領域に配置されないので全体検索は遅くなるが
追加，削除の操作で他の部分をずらす必要がなくなる
削除した要素の場所は，別の変数が使うこともできる
動的に増減する場合や，並び替えをするのに配列より内部処理が簡単となる

参考：検索・逆順で検索に向いたリスト

後ろからたどって探したい場合もある（スライスの[::-1]など）

位置	値	次	前
a	東京	b	無
b	大宮	c	a
c	長野	d	b
d	金沢	無	c

長野と金沢の間に富山を入れたい

pythonでは，金沢を探してその位置に
富山を入れれば良い

```
a.insert( a.index("金沢"), "富山")
```

内部ではどのような処理をすればよいか

後ろ側への接続のみのリスト(単方向リスト,方方向リスト)
後ろと前の接続を持つリスト(双方向リスト)

プログラム言語の仕様は構文記法と機能のみであるが
大量のデータを扱う場合は，性能などを考慮した実装をしたものを使う
(Cpython, Jython, pypy, メーカー提供，追加パッケージ，データ型)

スタック(stack)，キュー(queue)

スタック，キューは大きさが可変となる場合に重要なデータ構造

キュー　先に入れたものを先に取り出す(FIFO: First In First Out)

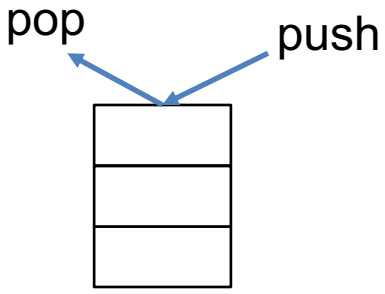
enqueue キューの最後に追加する(キュー長インクリメント)

dequeue キューの最初を取り出す(キュー長デクリメント)

スタック 後に入れたものを先に取り出す(LIFO: Last In First Out)

push　スタックに積む(スタック長インクリメント)

pop　スタックの先頭を取り出す（スタック長デクリメント）



キュー　待ち行列処理，サービス

スタック 前回の目次レベル，関数呼び出しなどで使われる

構造探索では，キューとスタックを使用するものが多い

スタック，キューとも長さが動的に変わる

enque, deque, push, popはプログラミング言語全般で使う機能の名称

先の双方向リストである順に並べたものをdeque(duble-ended queue)と呼ぶことがあり混同しないこと

pythonでのスタック，キューの実装

長さが変えられ，順番を保持するデータ構造　＝　リスト

キュー　que = list()
enqueue 後ろに追加すれば良いので　　que.append(追加するもの)
dequeue 前から取り出すので　　x = que.pop(0)
　　ここで，　x に先頭のものが入り，リストqueは0番目がなくなる
　　x = que[0]; que = que[1:] と同じ

スタック stk = list()
push　後ろに追加すれば良いので　　stk.append(追加するもの)
pop　最後から取り出すので　　x = stk.pop(-1)
　　ここで，　xに先頭のものが入り，リストstkは最後がなくなる
　　x = stk[-1]; stk = stk[:-1]と同じ

結局pythonでは，　リストの最後に追加して
最初を取ればキューに，最後を取ればスタック　を実現できる

無名関数(匿名関数, anonymous function)

ちょっとした関数を使いたいがいちいち定義するほどでもない
その場だけで有効な関数

辞書の値でソートするとき出てきたlambda が無名関数を作る

```
sorted(dict, lambda x: x[1])
```

関数型言語のλ式で使われていたが今は多くの言語で使える

書式　lambda 引数並び　: 引数を使った式

`lambda x : x*x` は　引数xの2乗を返す無名関数
`lambda x,y : x<y` はxが小さければTrueを返す

ソート関数sortedの例では第２引数に無名関数を渡していた！

関数に関数を渡す(高階関数)

関数は決まったことを行うが，場合により処理を変えたいことがある

関数に関数を渡す簡単な例　`math.sin(x)` とすると関数の結果を渡すので違いに注意

```
import math
def f1(f, x): return f(x)  # fは関数が入っている
f1(math.sin, math.pi) ⇨  1.2246467991473532e-16
f1(math.cos, math.pi) ⇨  -1.0
```

関数f1の第1引数は関数が渡されている(math.sin, math.cos)
f1で 引数1(引数2) を実行して結果を返している
math.sin, math.cosは　mathパッケージ内に関数の実体がある

sorted(d.items(), key=lambda x: x[1])は
第1引数に辞書(キーと値のペアのタプル）を
第2引数にソートする項目を求める無名関数(関数定義）を渡している
辞書は(key,value)のタプルが1要素で　無名関数の引数xとなる
x[1]はvalue すなわち，値でソートする指示をしている

map, recude, filter

C言語などでは，配列に値を入れてforループで同じ処理をする

```
# 整数の2乗数列 pythonのfor
x2=list()
for x in range(10):
    x2.append(x*x)
x2 ⇨ [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

リストの各要素を2乗する（リスト内包）

```
[x*x for x in range(10)]
```

map関数の記法 map(無名関数, iterable) # iterableは反復の意味

```
map(lambda x:x*x, range(10))
⇨ <map at 0x111ad37f0>  mapオブジェクトが戻り値が得られない)
```

値を確認

```
for i in map(lambda x:x*x, range(10)):
    print(i)
```

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
0000000000000000000000000000000000000000000000)
```

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

どの位時間がかかるか確認する

```
import datetime
def loop(n):
    t0 = datetime.datetime.now()
    for i in range(n):
        pass
    return datetime.datetime.now()-t0

loop(10*5)    ⇨ datetime.timedelta(0, 0, 2311)
loop(10**6)   ⇨ datetime.timedelta(0, 0, 25769)
loop(10**7)   ⇨ datetime.timedelta(0, 0, 235163)
loop(10**8)   ⇨ datetime.timedelta(0, 2, 326548)
loop(10**9)   ⇨ datetime.timedelta(0, 23, 240509)
loop(10**10)  ⇨ datetime.timedelta(0, 244, 989108)
```

10**100 ではどの位かかってしまうのか

↑
秒

遅延評価を行うもう一つの問題：
キー入力やネットワークを通じた通信などはいつデータがくるかわからない.

map処理 全ての要素に同じ操作をする

数列1から10の2乗を表示

```
for i in map(lambda x:x*x, range(10)): print(i, end=' ')
```

上記を変更せよ

1) 奇数ならTrueを，偶数ならFalseを返す

2) 3を足した値を返す

3) 10進数3桁の文字列にして返す

4) 辞書dict(a=1, b=2, c=3)をmap処理して"キー:値"を出力するmap処理を考えよ

上記1)から4)をリスト内包で書け

for文を2重にして掛け算九九を出力するリスト内包コードを考えよ

関数の引数について考える

```
#正弦表 loopで1つずつ計算
import math
y = list()
for i in range(-180, 180, 5):
    y.append( math.sin( math.radians(i) ) )

#正弦表 まとめて渡してまとめて結果を得る
import numpy
x = numpy.arange( -numpy.pi, numpy.pi, numpy.pi/36)
y = numpy.sin(x)
```

math.sinの引数は正数，実数，複素数

numpy.sinの引数はリスト(正確にはarray型)

nunmpy.sinはmap処理と同様なことを行っている．

import numpyでエラーとなる場合は，次のページ参照

追加パッケージインストール

numpyはpythonの標準パッケージではない(anacondaには含まれている)

パッケージを追加インストールするには端末アプリケーションで
pip install -U パッケージ名↵
pip自体が古くなると上記が失敗するので先に以下を行う
pip install -U pip↵

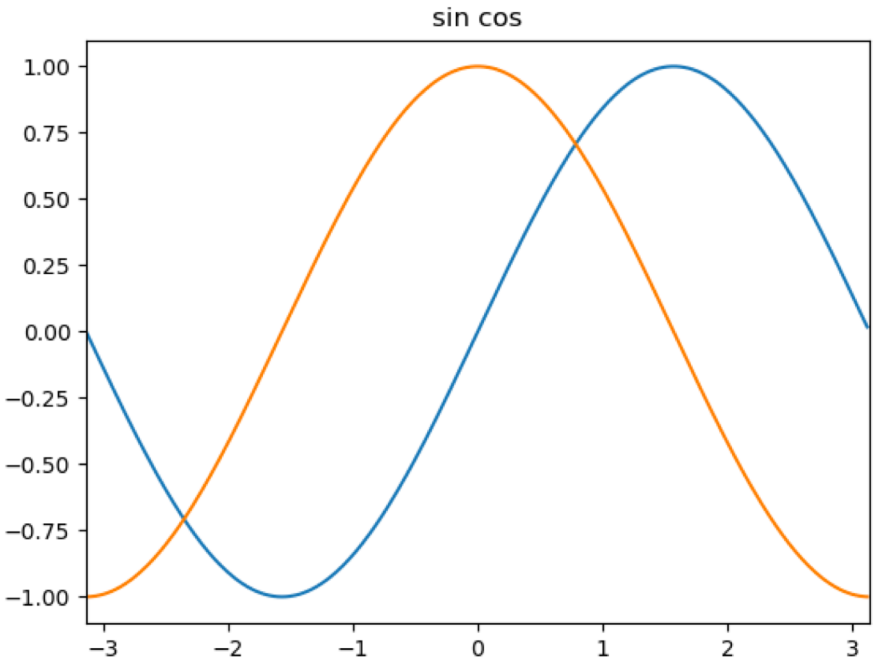
学内からはproxyサーバを指定する
pip --proxy=wwwproxy.kanazawa-it.ac.jp:8080 install -U パッケージ名↵

anacodaをインストーしている場合は，pipの前にcondaでインストール
conda install パッケージ名↵
anacondaのproxy設定は設定ファイル.condarcを編集して設定する
condaが古いというエラーがでたら
conda update conda↵
インストールしてある全てのパッケージはパッケージ名を　`--all` にする

パッケージを追加してグラフを描く

numpyとmatplotlibを追加インストールしておく

```
import numpy as N, matplotlib.pyplot as P
x  = N.arange(-N.pi, N.pi, N.pi/180)
P.plot(x, N.sin(x))
P.plot(x, N.cos(x))
P.xlim(-N.pi, N.pi)
P.title("sin cos")
P.show()
```



numpy.sin()は複数要素の正弦を返す
ループを使わなくて済む

reduce　集約　最初のn個を処理し，その結果と次を同じ演算を行うことを繰り返す

```
import functools as ft
ft.reduce(lambda x,y : x+y, range(1,11)) # 総和
55
ft.reduce(lambda x,y : x*y, range(1,11)) # 総積
362880

ft.reduce(lambda x,y : x+y, 'kanazawa')  # 正順
'kanazawa'
ft.reduce(lambda x,y : y+x, 'kanazawa')  # 逆順
'awazanak'
```

reduce()はfunctoolsパッケージをinportしないと使えない
総和は　sum(range(1,11)) でリストをsum関数に渡すことで得られる
総積は　numpyパッケージのprod関数でも行える
文字列を逆順にするにはスライスで可能

filter　条件に合うものを残す

```
for i in filter(lambda x:x%2==0,range(10)):
    print(i, end=' ')
```

```
0 2 4 6 8
```

リスト内包でも書ける

```
[x for x in range(10) if x%2==0]
```

```
[0, 2, 4, 6, 8]
```

pythonではメソッドチェーンでフィルターを実装するパッケージが多い
sqlデータベースをsql文を使わずアクセスするパッケージsqlalchemy

```
users = session.query(user.id, user.name).¥
    filter(user.name=='Tom').limit(10).order_by(user.id)
```

は
sql文の

```
select id, name from user where name='Tom' limit 10 order by id;
```

と同じ

map, reduce, filter

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文などで取り出さないと値が帰らない(遅延評価)

リスト内包記法や関数で同等の記述が可能

```
[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
```

上記は組み合わせられる

```
sum( [x*2 for x in range(11) if x%2==0] )
# [0,1,2,3,4,5,6,7,8,9,10] ⇨ [0,2,4,6,8,10] ⇨
# [0,4,8,12,16,20] ⇨ 60
```

リスト以外の内包

```
[[0]*3]*2 # 内包ではないが　2行3列のリストを返す　0行列
['%3d'%x for x in range(5)] # 数列を書式整形したリストを返す
[[x*y for x in range(1,4)] for y in range(1,3)]  # 2重
```

辞書の内包

```
key = ['a', 'b', 'c']    # キーのリスト
val = [1, 2, 3]          # 値のリスト
{k:v for k, v in zip(key,val)} # 2つから辞書を作る
{'a': 1, 'b': 2, 'c': 3}
```

```
# 値が奇数なら2倍する．偶数なら捨てる
{k:v*2 for k, v in zip(key,val) if v%2==1}
{'a': 2, 'c': 6}
```

演習

今回の例題を確認せよ

sin-cosグラフの変更

例では　numpyパッケージのsin(),cos()を使用している
numpy.sin(), numpy.cos()はリストを引数で受け取り一気に結果を返す

mathパッケージのsin(),cos()を使用する
map関数，lambdaを使用し渡されたリストの各要素を同様の処理をする

x軸の値を作成するのにnumpy.arange()を使用して良い

レポート1実施

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

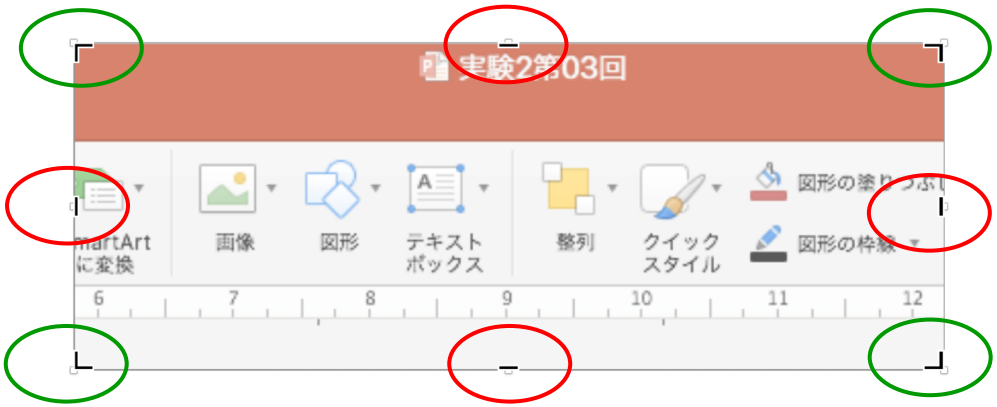
他者にわかりやすいように書く
小さくて字が読めない場合や画像に余白が多い場合は減点する

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

ハードコピー(スクリーンショット）の撮り方

OSにより操作方法は異なるが
Windows系では,　ALTキーとprtscキーを同時に押すと
一番最後に使った(最前面)のウインドウがクリップボードに入る
Word等で貼り付け(Ctrl-V)を行えば画像が入れられる

図のトリミング
画像の四隅に不要な部分があればトリミングする
画像を選択して右クリック⇨トリミング
下図の赤丸部分に出る棒線(カーソル)をドラッグして不要な部分を省く



図の拡大・縮小
左図の４隅（緑丸部分）をドラッグする
縦横比は変えないこと
(斜め方向にドラッグすればよい)