コンピュータサイエンス入門第一

永藤 直行

3rd quarter

Part I

Prologue

Prologue

Outline

科目の概要

- 学期: 水曜日 3-4 限
- 場所: 南 4 号館 3 階第 1 演習室か W631 教室
- 担当教員: 永藤 直行 (ナガトウ ナオユキ)
- 連絡先: nagatou@presystems.xyz
- 質問時間: 講義終了後かメイルで
- CS4b クラスのサイト: OCW-i か
  https://sites.google.com/a/presystems.xyz/sample/home/elementary-computer-science
- 共通サイト: 共通サイト

評価基準

- 講義は全 6 回，期末試験 1 回
- 宿題: 3 回 $3 \times 5 = 15$ 点
- レポート: 3 回 $15 + 15 + 20 = 50$ 点
- 期末試験: 1 回 35 点
- 課題提出:
  - 講義時間中に課題を出します
  - 提出方法はその都度指定します
- 課題提出で出欠確認に変えます

講義日程

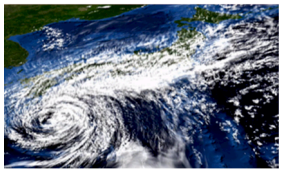
- 教室を間違えないように
- 第 3 回目 10 月 16 日 (Wed) は休講

回	題目	場所
1	ガイダンス，計算の基礎	南 4 号館 3 階第 1 演習室
2	プログラミング演習	南 4 号館 3 階第 1 演習室
3	配列，文字列	南 4 号館 3 階第 1 演習室
4	プログラミング演習	南 4 号館 3 階第 1 演習室
5	暗号入門	南 4 号館 3 階第 1 演習室
6	プログラミング演習	南 4 号館 3 階第 1 演習室
7	試験	西 2 号館 W631

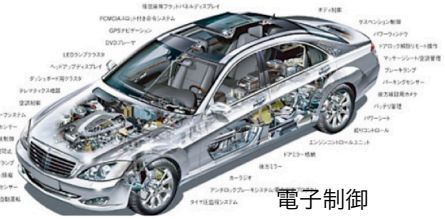
Outline

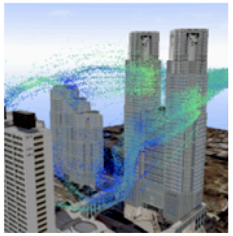
講義の目標

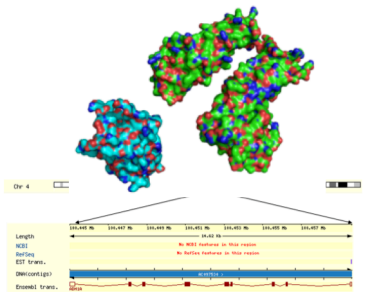
- 本講義では，このコンピュータサイエンスの基本をなす考え方を，課題をやることを通して体得する
- 物理現象をシミュレートしたり
- 経済活動にともなう帳票類を管理したり
- 機器を制御したり
- コンピュータがいろいろな場面で利用されている



スパコン


電子制御


シミュレーション



帳票管理

コンピュータに載せるとは?

- なぜコンピュータが利用できるのか
- どうやってコンピュータに載せるのか

コンピュータに載せる

やりたいことを計算をもちいて表現し，コンピュータに処理させること

目標

- CS 第一
  - 計算で表現するとは何か
  - コンピュータで処理するとは
- CS 第二 第二の講義概要
  - 計算の強力な道具 ⇒ 再帰
  - 載せ方の上手下手があること ⇒ アルゴリズムやデータ

講義内容

- 以下の演習を通して実感しながら理解していく

演習内容

- 演習課題 1: 四則演算でアニメーション
  - 計算の基本要素を知る
- 演習課題 2: 循環小数
  - 配列とは
- 演習課題 3: 暗号解読に挑戦
  - プログラミングとは
  - 計算の組み立て方

Part II

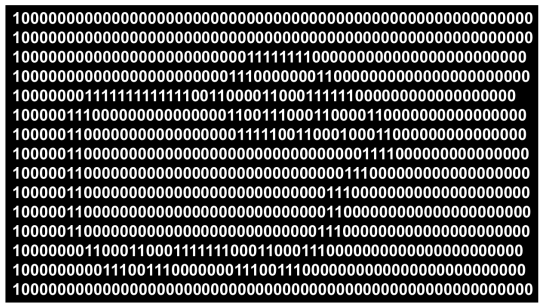
計算の基本

計算の基本

Outline

課題 1 の目標とテーマ

- 目標
  - 計算の基本要素を知る
- テーマ
  - 四則演算でアニメーション
  - ひつじさん



CS のこころ

- しつこいようですが，すべては計算
- コンピュータに載せるには
  - 対象をデータとして表すこと
  - 処理を基本演算の組み合わせてで表すこと
- 処理とはコンピュータのなかの抽象的な世界に存在して
- データというもう一つの抽象的な存在を操作する
- この処理やデータをプログラミング言語の記号をもちいて注意深く構成したのがプログラム

Outline

はじめに
課題 1 の目標とテーマ
CS のこころ
データは数である
Bit と Byte
自然数の n 進表記
文字データ
画像と音
計算 = ±1 と繰り返し
計算とは
計算の基本要素
プログラムの書き方
プログラミングのための仕掛け
宿題 1 を動かしてみる
Terminal について
ソースファイルの編集
プログラムを走らせてみる

データは数である

- データから見ていくことにして
- データはすべて 2 進列で表される
  - 自然数，整数，実数: 18, -3, 3.14 など
  - 文字: 文字コード: ASCII, Unicode など
  - 画像，映像
  - 音
  - におい，味，触覚

情報とは

- ここで情報とは何か考えてみます
- 情報とは"ある物事，事情についてのお知らせ"
- 情報の価値はどう決まるか?
  - 驚きをもって受け止められる情報は価値が高い?
  - 日常的な情報は価値が低い?

まずは情報量というもの

1. ある結果や情報を得る場合を考える
2. 結果や情報を生じる事象が確率現象であると見なす
3. 確率 p の事象の情報量を $I(p)$ であらわす
4. $I(p)$ は単調減少関数
   - 頻繁に起こっていること (p が大きい) は情報量が少ない)
   - 頻繁に起こらないこと (p が小さい) は情報量が多い)
5. 連続関数である
   - 確率のわずかな変化で情報量が大きく変化するのは不自然

情報量の定義

ある事象 a の生起確率を p_a とするとその情報量 $I(p_a)$ は $\log_2 \frac{1}{p_a} = -\log_2 p_a$ であらわすことにする

ビットとは

1 bit とは

1. 今，２つの事象を考える
2. 同じ確率 $P = \frac{1}{2}$ で生起するとする
3. このとき，ひとつの事象 a の情報量 $I(a) = \log_2 \frac{1}{\frac{1}{2}} = 1$
4. これが 1 bit
5. 確率 $\frac{1}{2}$ で起こる事象を知った時の情報量が 1 bit(ビット)

- それでは確率 $\frac{1}{10}$ で起こる事象を知った時は 1 hartley(ハートレー)
- 確率 $\frac{1}{e}$ では 1 nat(ナット)

情報の記録

- 情報はビットの列として記録
- 明確に区別された 2 つの状態で記録しています
  - 磁性体の向き，電圧の高低，スイッチの開閉
- 計算機科学では 2 つの状態を便宜的に 0 と 1 として議論しています

ビットによる表現

- 2 つの状態を取り得るデバイスを N 個並べてそれぞれ独立としたらどれだけの情報があらわせるか
- 答えは $\log_2 2^N = N$ となり N ビットの情報量となります
- N ビットでどれだけの事象を区別できるでしょうか
- 答えは 2 個の要素から N 個の重複順列 ${}_2\Pi_N = 2^N$ です

バイトとは

- 人間にとって意味をなす長さ N の小ブロック
- 現在のコンピュータでは 8 bit としています

数表記

- ビットの列で表すことは先に述べました
- では自然数はどうあらわすでしょう
- 数表記は 10 進が唯一の方法ではありません
- n 進表記が可能です
- 数はどうコンピュータ内で表現されるかみていきます

n 進表記

- 実は日常的に n 進法を利用しています
- 時間は 24 進法，60 進法，30 進法，360 進法をもちいています
- たとえば 24 進法では 24 になったら位が一つ上がります
- コンピュータでは 2 進法をもちいて自然数を表します
- 2 進法ではやはり人間には分かりずらいので 8 進法や 16 進法であらわすことが多いです

n 進法の各桁

- 2 進法，8 進法，16 進法でも 10 進法と同じように位どりによってあらわします
- 良くご存知のように 10 進法では 1 桁を 0-9 のいづれかであらわしています
- 123 という自然数であれば $1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$ といった具合です

2 進法の各桁

- 2 進法では各桁は 0 と 1 だけになります
- 10 進法の場合と同様に位どりします，ただし底が 2 になります
- $(010)_2$ という自然数であれば $0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$ といった具合です
- この例では $(2)_{10}$ は位が一つ上がって $(010)_2$ となっています

16 進法の各桁

- 2 進法では桁が多くなって見ずらいので 16 進で表記します
- 16 進法では各桁:
  - 0,1,2,3,4,5,6,7,8,9 と 0-9 までは 10 進と同じ
  - 10,11,12,13,14,15 は A,B,C,D,E,F をもちいます
- 10 進法の場合と同様に位どりします，ただし底が 16 になります
- $(1F0)_{16}$ という自然数であれば $1 \times 16^2 + F \times 16^1 + 0 \times 16^0$ といった具合です

各数字の対応

10 進	8 進	16 進	2 進
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111

10 進	8 進	16 進	2 進
8	10	8	1000
9	11	9	1001
10	12	A	1010
11	13	B	1011
12	14	C	1100
13	15	D	1101
14	16	E	1110
15	17	F	1111

n 進数の変換

- m 進数から n 進数への変換
- 手始めに 10 進数から 2 進数への変換

Example (10 進 ⇔2 進)

$$\begin{array}{l} 2)19\cdots 1 \\ \hline 2)9\cdots 1 \\ \hline 2)4\cdots 0 \\ \hline 2)2\cdots 0 \\ \hline 2)1\cdots 1 \\ \hline 0 \end{array} \quad \begin{array}{l} \text{Low} \\ \\ \\ \\ \\ \text{High} \end{array} \qquad \begin{array}{r} 1\times 2^0 = 1 \\ \hline 1\times 2^1 = 2 \\ \hline 0\times 2^2 = 0 \\ \hline 0\times 2^3 = 0 \\ \hline 1\times 2^4 = 16 \\ \hline 19 \end{array}$$

Quiz: 10 進表記 ⇔ 2 進表記 の変換

Quiz

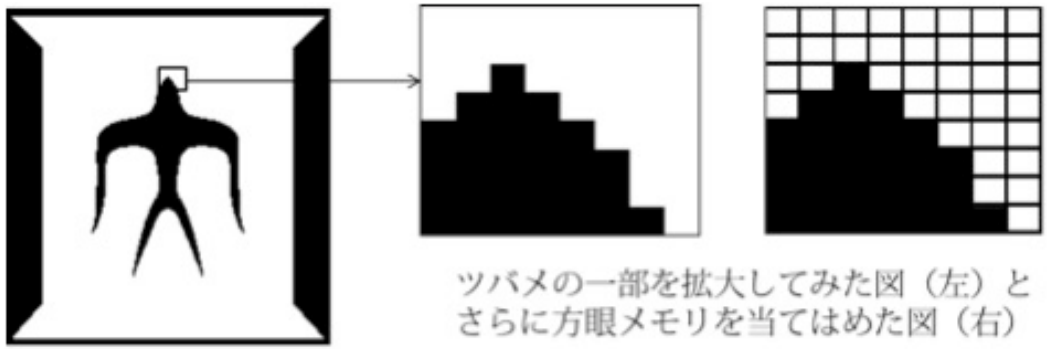
1. $131_{10}, 112_{10}$ を 2 進数に変換してみてください
2. 1 で得られた 2 進数を 10 進表記に戻してください
3. もとの 10 進数が得られれば正しく変換できています
4. この数字は東工大に割り当てられた IP アドレスになります

文字データの表現

- コンピュータは数値だけでなく文字も処理することができる
- 文字をコード化して処理する
- 詳細は第 3 回目で

画像

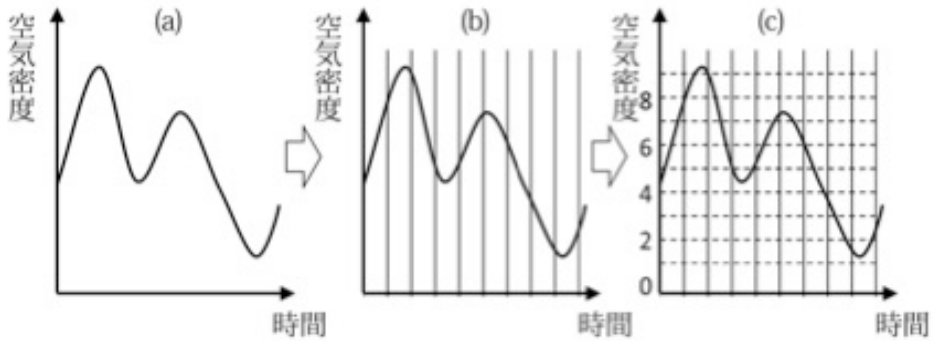
- つぎは画像データについて見てみます
- 画像も二進列で表せます
- ビットマップというファイル形式
  - マス目にわけ，白い部分を 1，黒い部分を 0 としてビットの列を作る



ツバメの一部を拡大してみた図（左）と
さらに方眼メモリを当てはめた図（右）

音

- つぎは音データ
- 波の符号化
  - (a) は波形
  - (b) は標本化
  - (c) はそれぞれの標本値
  - この標本値を順番にならべた二進列
- 音符の符号化 (Standard MIDI)
  - 音符やテンポを符号化
  - .mid



Outline

はじめに
課題 1 の目標とテーマ
CS のこころ
データは数である
Bit と Byte
自然数の n 進表記
文字データ
画像と音
計算 = ±1 と繰り返し
計算とは
計算の基本要素
プログラムの書き方
プログラミングのための仕掛け
宿題 1 を動かしてみる
Terminal について
ソースファイルの編集
プログラムを走らせてみる

計算とは

- つぎは計算について
- 計算には入力と出力があって
- 入力と出力の関係をとくに関数とよぶ
- この場合，ある入力に対して出力は一意に決まる
- また関数には適当な名前をつけることができる
- 関数を合成してより複雑な計算を実現していく
- ここまではまだ入出力の関係としか云っていない

Example (最大公約数)

最大公約数 $gcd(n, m)$ は n, m の公約数で最大ものと定義されるが，どう求めるか方法は書いていない

計算の方法

- 計算の方法について考える
- 計算機科学では関数といった時はこちらの意味
- 計算の方法のことをアルゴリズム (algorithm) といっている
- アルゴリズムとデータを特定の形式で書いたものがプログラム

ソースコード 1: 最大公約数

```
1 # Greatest common divisor
2 # Input: 自然数 x, y
3 # Output: gcd(x,y)
4 ###
5
6 def gcd(x,y):
7   ans=1
8   n=min(x,y)
9   for i in range(1,n):
10    if (x%i==0) and (y%i==0):
11      ans=i
12  return (ans)
13 print(gcd(x,y))
```

計算の基本要素
計算 = ±1 と繰り返し

- 合成していくとして最も基本となるものは何か
- 結論から云ってしまえば，ある値に ±1 する操作と繰り返しと条件分岐である

ソースコード 2: 加算

```
 1 # add.py
 2 # Input: 自然数 a, b
 3 # Output: a + b
 4 ###
 5
 6 a = int(input("? "))  # 入力された自然数を a に代入
 7 b = int(input("? "))  # 入力された自然数を b に代入
 8 wa = a                # a の値を wa に代入
 9 while b > 0:          # b が 0 より大きい間は end までを繰り返す
10   wa = wa + 1        #   wa + 1 の値を wa に代入
11   b = b - 1          #   b - 1 の値を b に代入
12 print(wa)            # wa の値を出力
```

基本要素だけの乗算

ソースコード 3: 乗算

```
 1 # mult_basiconly.py
 2 # Input: 自然数 x, y
 3 # Output: x * y
 4 ###
 5
 6 x = int(input("x? "))          # 入力された自然数を x に代入
 7 y = int(input("y? "))          # 入力された自然数を y に代入
 8 seki = 0                       # seki を 0 で初期化
 9 while y > 0:                   # y が 0 より大きい間は end までを繰り返す
10   a = seki
11   b = x
12   wa = a                       # 和のプログラム add.py を挿入
13   while b > 0:
14     wa = wa + 1
15     b = b - 1
16   seki = wa                    # wa の値 (seki + x) を seki に代入
17   y = y - 1                    # y - 1 の値を y に代入
18 print(seki)                    # seki の値を出力
```

Back to composit-slide Back to while-slide

Outline

はじめに
課題 1 の目標とテーマ
CS のこころ
データは数である
Bit と Byte
自然数の n 進表記
文字データ
画像と音
計算 = ±1 と繰り返し
計算とは
計算の基本要素
プログラムの書き方
プログラミングのための仕掛け
宿題 1 を動かしてみる
Terminal について
ソースファイルの編集
プログラムを走らせてみる

プログラミングのための仕掛け

- プログラミング言語にはプログラミングのための 3 つの仕掛けがある
  - 基本式: プログラミングに関わる最も単純なもの
  - 組合せ法: 単純なものからより複雑なものをつくる (構文として定義されている)
  - 抽象化法: 合成物に名前をつけ基本式と同様に扱う
- データについても同様

Python における基本式

- 式の基本要素は 自然数，整数，実数，虚数などがある
  - 自然数の例: 286, 386, 486
  - 講義では自然数だけあつかうので注意
- 基本要素と演算子 +, −, *, //, %, ** などを組み合わせて式をつくる
  - 自然数の例: 286 + 386, 486
  - 入れ子にできます: 286 + (386 + 486)
- 式には名前をつけることができ，この名前のことを変数と呼でいます
- その名前で参照することもできます
  - 例: abc = 286 + 386
  - 例: efg = abc + 486
- = は論理記号ではなくて代入をあらわすので注意
- abc は a × b × c ではなく変数名なので注意

Python における合成

- 式を順番に並べる
- 実行は上から下へ，左から右に順番に実行される
- 前に実行された式の結果は変数をもちいて参照
- 実行順序を変えたいときは while や if を使う
- Jump to an example

if 文

- 条件によって実行順序を変更する
- Jump to an example
- 条件式は Python ドキュメント 6.10, 6.11, 6.12 節参照

while 文

- 実行列を繰り返し実行する
- Jump to an example

±1 だけで四則演算を作る
宿題 1

- 宿題 1 を完成して実行してみてください
- https://sites.google.com/a/presystems.xyz/sample/home/elementary-computer-science に div-skeleton.py と sub-skeleton.py があるはずなのでそれを完成させてくだい
- 次回講義までに OCW-i か紙で提出

ソースコード 4: sub.py

```
1  # sub.py
2  # Input: 自然数 a, b
3  # Output: a - b
4
5  a = int(input("a? "))
6  b = int(input("b? "))
7  sa = a
8  while    :
9    sa =
10   b =
11 print(sa)
```

ソースコード 5: div.py

```
1  # div.py
2  # Input: 自然数 x, y
3  # Output: x ÷ y の商と余り
4
5  x = int(input("x? "))
6  y = int(input("y? "))
7  shou =
8  amari =
9  while    :
10   shou =
11   amari = amari - y
12 print(shou)
13 print(amari)
```

Terminal の起動

- Launchpad からターミナルのアイコンをクリックして起動
- これで shell が起動しファイル，ディレクトリの操作とプログラムの起動，終了ができます

ファイルに対する操作

操作	コマンド	実行例	
生成	touch	touch *name*	指定した名前で空のファイルを生成
名前変更	mv	mv *oldfile newfile*	*oldfile* という名前を *newfile* という名前に変更
複製	cp	cp *srcfile dstfile*	*srcfile* を複製して *dstfile* という名前をつける
表示	less	less *name*	*name* の内容を表示
消去	rm	rm *name*	指定した名前のファイルを消去

ディレクトリに対する操作

操作	コマンド	実行例	
作成	mkdir	mkdir *name*	指定した名前で空のディレクトリを生成
一覧	ls	ls *dir*	*dir* の中身一覧を表示
格納	mv	mv *file dir*	*file* を消去して *dir* に格納
	cp	cp *file dir*	*file* を複製して *dir* に格納
名称変更	mv	mv *olddir newdir*	*olddir* を *newdir* に変更
消去	rmdir	rmdir *dir*	空の時には *dir* を消去
	rm	rm -r *dir*	中身ごと *dir* を消去
移動	cd	cd *dir*	作業ディレクトリを移動
	pushd	pushd *dir*	作業ディレクトリを移動して現在のディレクトリを保存
	popd	popd *dir*	作業ディレクトリを保存したディレクトリに移動
表示	pwd	pwd	作業ディレクトリを表示
	dirs	dirs	保存したディレクトリを表示

ターミナルをつかってみよう

- ターミナルを使って課題の準備をします
- Terminal を起動
- コマンドプロンプト > が表示されたらホームディレクトリの下に適当なディレクトリ (ここでは CS1/kadai) を作成し
- https://sites.google.com/a/presystems.xyz/sample/home/elementary-computer-science から必要なファイルを作成したディレクトリにダウンロード

課題準備

```
> mkdir CS1/kadai # 課題用のディレクトリを作成
> cd CS1/kadai    # 課題用ディレクトリに移動
> cp sub-skeleton.py sub.py # 課題をコピー
```

ソースファイルの編集

- テキストエディタと呼ばれるソフトウェアを使って編集
  - vim, emacs など
- ターミナルからコマンド入力して起動

editor の起動

```
> open -a Macvim sub.py
あるいは
> open -a Emacs sub.py
```

プログラムを走らせてみる

- コマンドプロンプト > が表示されたら phtyon3 とソースファイル名と入力して return
- Python プログラムが実行されます

Python の起動

```
> python3 sub.py
```

Part III

CS 第 1—課題 1

CS 第 1 —課題 1

Outline

課題 1 の説明
四則演算でアニメーション

- 課題: 四則演算でアニメーションを作成してください
  - 動きがあること
  - 計算だけで動かすこと
- 提出物:
  - 作成したアニメーションプログラムのソースコード (anime.py)
  - 作成したアニメーションプログラムの使い方の説明
  - 作成したアニメーションプログラムの計算の仕組みの説明
    - 工夫した点も書くこと
- 採点者はソースコードを読まないと仮定して説明すること
- Python 言語の説明は不要
- 期限は来週のこの時間まで

ひつじさんを動かしてみる

- 大きな数字を定義して
- 各桁をひとつの画素とみなす
- 各桁は 0–9 でこの違いで絵にする
- 14 個の自然数
- 先頭は 1 にしないとずれる
- 動かすときは桁をシフトさせて (ここでは 10 で割っている) 動かす

ソースコード 6: sheep.py (declaration)

```
 1 ################
 2 # pattern def. #
 3 ################
 4 d0  = 10000000000000000000000000000000000000000000000000000000000000
 5 d1  = 10000000000000000000000000111111110000000000000000000000000000
 6 d2  = 10000000000000000000000001110000000110000000000000000000000000
 7 d3  = 10000000111111111111110011000011000111111000000000000000000000
 8 d4  = 10000011100000000000000011001110001100001100000000000000000000
 9 d5  = 10000011000000000000000001111100110001000110000000000000000000
10 d6  = 10000011000000000000000000000000000001111000000000000000000000
11 d7  = 10000011000000000000000000000000000011100000000000000000000000
12 d8  = 10000011000000000000000000000000000111000000000000000000000000
13 d9  = 10000011000000000000000000000000001100000000000000000000000000
14 d10 = 10000011000000000000000000000000011100000000000000000000000000
15 d11 = 10000000110001100011111111000110001110000000000000000000000000
16 d12 = 10000000000111001110000000111001110000000000000000000000000000
17 d13 = 10000000000000000000000000000000000000000000000000000000000000
```

ひつじさんを動かしてみる

- 動かすときは桁をシフトさせて (ここでは 10 で割っている) 動かす
- a0 を足しているのは絵がずれないようにするため

ソースコード 7: sheep.py (shift)

```
18      # shift #
19      a0 = d0
20
21      a1  = (a1 -  a0) // 10 + a0
22      a2  = (a2  - a0) // 10 + a0
23      a3  = (a3  - a0) // 10 + a0
24      a4  = (a4  - a0) // 10 + a0
25      a5  = (a5  - a0) // 10 + a0
26      a6  = (a6  - a0) // 10 + a0
27      a7  = (a7  - a0) // 10 + a0
28      a8  = (a8  - a0) // 10 + a0
29      a9  = (a9  - a0) // 10 + a0
30      a10 = (a10 - a0) // 10 + a0
31      a11 = (a11 - a0) // 10 + a0
32      a12 = (a12 - a0) // 10 + a0
33      a13 = (a13 - a0) // 10 + a0
34      a14 = (a14 - a0) // 10 + a0
35      a15 = (a15 - a0) // 10 + a0
36      a16 = (a16 - a0) // 10 + a0
37      a17 = (a17 - a0) // 10 + a0
38      a18 = (a18 - a0) // 10 + a0
39      a19 = (a19 - a0) // 10 + a0
40      a20 = (a20 - a0) // 10 + a0
```

Part IV

配列

配列

Outline

配列

- データオブジェクトの順序づけられた集まり
- 配列の各オブジェクトを配列の要素と呼びます
- 各要素は 0 から始まる自然数に対応付けられていて
- 自然数のことをインデックスと呼んでいます
- 各要素は配列名 [インデックス] で参照することができます

ソースコード 8: 単純な変数

```
1 d1 =  10000000000000000000000000000000
2 d2 =  10000000001100001100000000000000
3 d3 =  10000000001100001100000000000000
4 d4 =  10000000000000000000000000000000
5 d5 =  10000011000000000000011000000000
6 d6 =  10000001100000000000110000000000
7 d7 =  10000000110000000001100000000000
8 d8 =  10000000001111111110000000000000
9 d9 =  10000000000000000000000000000000
```

ソースコード 9: 配列

```
1  d = [10000000000000000000000000000000,
2       10000000001100001100000000000000,
3       10000000001100001100000000000000,
4       10000000000000000000000000000000,
5       10000011000000000000011000000000,
6       10000001100000000000110000000000,
7       10000000110000000001100000000000,
8       10000000001111111110000000000000,
9       10000000000000000000000000000000]
```

配列の参照と代入

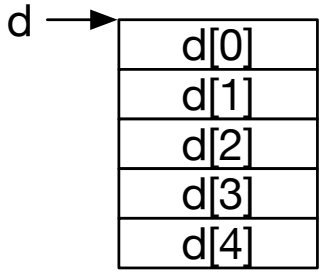
- 連続したメモリ領域を確保
- 先頭を 0 としてインデックスで参照 (ここは板書します)

ソースコード 10: 代入

```
1  d = [0]*9
2  d[0]=100000000000000000000000000000
3  d[1]=100000000011000011000000000000
4  d[2]=100000000011000011000000000000
5  d[3]=100000000000000000000000000000
6  d[4]=100000110000000000011000000000
7  d[5]=100000011000000000110000000000
8  d[6]=100000001100000000110000000000
9  d[7]=100000000011111111000000000000
10 d[8]=100000000000000000000000000000
```

ソースコード 11: 参照

```
1  for i in range(9):
2    print(d[i])
```



配列の例

- 総和を求める
- ソースコード 12: 整数の配列を宣言する例
- ソースコード 13: 入力した整数を配列にする例

ソースコード 12: 総和

```
1 a=[2,4,6,8,10,12]
2 s=0
3 for k in range(len(a)):
4   s=s+a[k]
5   k=k+1
6 print(s)
```

ソースコード 13: 入力した数の総和

```
1 a = list(map(int,input("numbers? ").split()))
2 s=0
3 for k in range(len(a)):
4   s=s+a[k]
5   k=k+1
6 print(s)
```

最大値を求める
宿題 2

- https://sites.google.com/a/presystems.xyz/sample/home/elementary-computer-science の max-skeleton.py を完成させる
- 最大値=max($a_1, a_2, \cdots, a_n$)
- `for j in range(0,n)` は j を 0 から n-1 まで繰り返すという意味

ソースコード 14: max.py

```
1 # max.py
2 # 入力: 整数の列
3 # 出力: 最大値
4 array = list(map(int,input("numbers? ").split
      ()))
5 if not array:  # array が空の場合
6         raise ValueError("...")  # 入力エラー
7 # 以下が計算部分
8 max_value = array[0]       # array[0] を一時的に
      最大値に
9 max_index = 0
10 for i in range(0,(len(array))):
11   if :                    # より大きな数を探す
12     max_value =
13     max_index =
14 print(max_value, max_index)
```

出力例

```
> python3 max.py
numbers? -3 8 19 -4
19 2
```

文字データの表現

- コンピュータは数値だけでなく文字も処理することができる
- 文字はコード化されて処理される
- 文字列は文字の配列

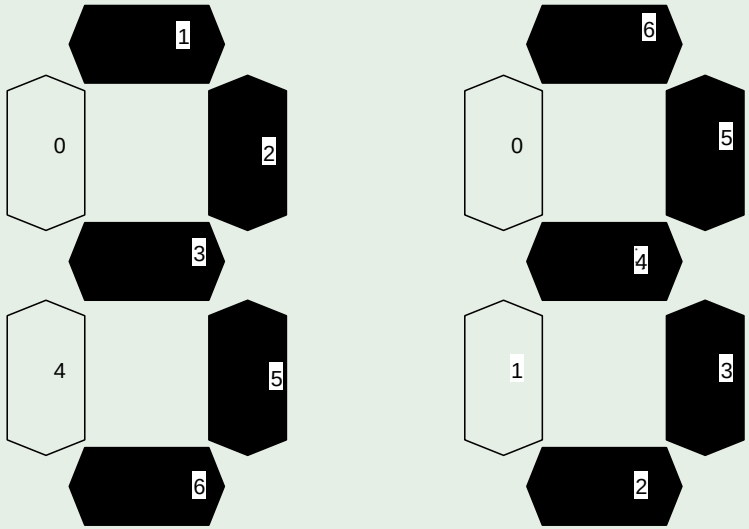
文字コード

- コードとは，文字や記号をコンピュータで扱うための記号です
- コンピュータ内では適当な正整数が文字や記号に割り振られています
  - 整数は 2 進数で表されているので 0 と 1 の列にコード化されます
- コードは任意に決めることもできます
- しかし，各コンピュータで違っていては不都合が生じます
- 異なるコンピュータでは全く違った文字になってしまうかも知れません
  - 文字コードが違っているとうまく表示できません

コードの違いの例

- 3 という文字のコードの例です
- 下の図は電光掲示板の例です
- 左の図では 110 1110$_{(2)}$ とコードを割り当てています
- 右の図では 111 1100$_{(2)}$ とコードを割り当てています
- このように違ったコードを対応づけることもできます

Example (電光掲示板の例)



ASCII コード

- 共通のコード体系として ASCII コードが策定されました
- これは英語のアルファベットと数字と記号にコードを割り当てています
  - ASCII Code Chart

日本語漢字のコード体系

- 現在は日本語など漢字圏の文字もコードが割り当てられています
- JIS, Shift-JIS, EUC, Unicode があります
  - これらは異なるコード体系です
  - 同じ文字でも異なるコードが割り当てられています
- 漢字圏で Unicode 以前に用いていたコードの規格
  - 日本: JIS X 0208-1990, JIS X 0212-1990(第一水準，第二水準，補助漢字)
  - 中国: GB 2312-80, GB 12345-90···
  - 台湾: CNS 111643-1986
  - 韓国: KS C 5601-1987, KS C 5657-1991
- Python3 は Unicode を利用しています

Python での文字列

- 文字列は各文字の配列として扱える
- str という名前の配列の各要素に一文字入っている
- 0 番目から順番にインデックスで参照できる

ソースコード 15: stringPrint.py

```
1 # stringPrint.py
2 # 文字列処理の練習プログラム
3 # 入力: 文字列
4 # 出力: 文字列の文字を 1 行 1 文字で出す
5
6 str = (input("strings? ")).encode("ascii
      ")
7 print(str)
8 for k in range(0,len(str)):
9    print(chr(str[k]), hex(str[k]))
```

出力例

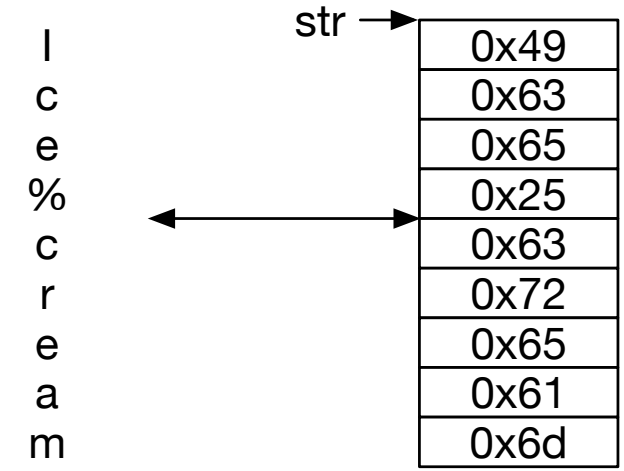
```
> python3 stringPrint.py
strings? Ice%cream
I 0x49
c 0x63
e 0x65
% 0x25
c 0x63
r 0x72
e 0x65
a 0x61
m 0x6d
```

文字列の表現

- 文字列は文字の配列
- 各文字はその符号 (文字コード) で表されて
- 各要素に各文字コードを格納

出力例

```
> python3 stringPrint.py
strings? Ice%cream
I 0x49
c 0x63
e 0x65
% 0x25
c 0x63
r 0x72
e 0x65
a 0x61
m 0x6d
```



英小文字だけを画面に出力
宿題 3

- https://sites.google.com/a/presystems.xyz/sample/home/elementary-computer-science の abcPrint-skeleton.py を完成させる

ソースコード 16: abcPrint.py

```
1 # abcPrint.py
2 # 文字列処理の練習プログラム，小文字だけ出力
3 # 入力: 文字列
4 # 出力: 文字列の文字で小文字のみ出力する
5 ss = (input("strings? ")).encode("ascii")
6 for k, code in enumerate(ss):
7   if :                        # 小文字ならば
8     print(chr(ss[k]))         # 文字を表示する
```

出力例

```
> python3 abcPrint.py
strings? Ice%cream
c
e
c
r
e
a
m
```

課題 2 予告

- 来週の予定です
- Jump to Quiz 2

Part V

CS 第 1—課題 2

CS 第 1 —課題 2

Outline

準備

1. 前回の宿題を実際に動かしてみます
2. Terminal を起動
3. > mkdir CShomework として宿題用ディレクトリを作成
4. abcPrint-skeleton.rb と max-skeleton.rb を宿題用ディレクトリにダウンロード
5. > cp abcPrint-skeleton.py abcPrint.py
6. > cp max-skeleton.py max.py とコピーして
7. 前回の宿題 2, 3 を完成させて実行してみる

宿題 2 の実行方法

```
> python3 max.py
numbers? -3 8 19 -4
19 2
```

宿題 3 の実行方法

```
> python3 abcPrint.py
strings? Ice%cream
c
e
c
r
e
a
m
```

Outline

課題 2 テーマ

- 前回までにみたように配列は複数のデータを統一的に扱う方法を提供した
- それ以外の例もこの課題で見ていくことにします

やってほしいこと

- 今，整数の組で表わされている有理数を小数表記に変換するプログラムを作ろうとしているとします
- 配列を使って循環小数になっても停止するようにプログラム junkan.py を改良してください
- ただし，分子は 1 に固定する

junkan.py

- まずは junkan.py を実行してみてください
- あまりは 0 から d-1 の有限の範囲なので，10 倍して割るを繰り返しているとどこかで同じあまりが出てくるはず

ソースコード 17: junkan.py

```
 1 # junkan.py
 2 # 配列の使い方の練習（循環小数を循環するまで求める）
 3 # 入力: d
 4 # 出力: 1/d の各桁を循環するまで求める
 5 d = int(input("1/d d(>=2)? "))
 6 print("1/",d," を求めます")
 7 x = 1
 8 print("0.", end=""),
 9 while (True):
10   x = x * 10
11   q = x // d
12   x = x % d
13   print(q, end="")
14   if x == 0:
15     break
16 print("")
```

Part VI

関数とサブルーチン

関数とサブルーチン

11 関数，メソッド，サブルーチン
- 関数宣言
- 計算の抽象化

12 関数を使って世の中の事象を抽象化
- 暗号方式のおはなし
- 暗号通信のプログラム
- 暗号解読 (Cryptoanalysis)

Outline

関数，メソッド，サブルーチン

- 抽象化の方法について触れてきた
  - 変数: 計算の結果に名前をつけるということ
  - 配列: データの集まりを名前をつけて抽象化
- 今回は計算を合成する抽象化について見てみます
- 合成したものに名前をつけて単一の手続きとして抽象化する方法

Python における合成手続き

- Python では def というキーワードをつかって定義します
- def のあとに関数名と仮引数を書いて，最後にコロン：(忘れずに)
- 仮引数は関数を呼び出したときに実引数にバインドされる
- 引数はその関数内だけで有効

ソースコード 18: add.py

```
1 a = int(input("a? "))
2 b = int(input("b? "))
3 wa = a
4 while b > 0:
5   wa = wa + 1
6   b = b - 1
7 print(wa)
```

ソースコード 19: 関数定義

```
1 def add(x,y):
2   wa = x
3   while y > 0:
4     wa = wa + 1
5     y = y - 1
6   return(wa)
```

ソースコード 20: 関数適用

```
7 def mult(x,y):
8   seki = 0
9   while y > 0:
10    b = x
11    seki = add(seki,b)
12    y = y - 1
13  return(seki)
```

ブラックボックス抽象としての関数

- “プログラムを部分に分ける”
- どう分けるかが重要になる
- 他のプログラムの部品として使え，まとまった仕事ができるようにわける
- 部品としての関数はどう計算するかには関心をもたず，計算結果にだけ関心を持てば良い

ソースコード 21: 集合演算

```
1 ### Set operations
2 def union(seta,setb,result):
3   global Size
4   for i in range(Size):
5     result[i]=seta[i] or setb[i]
6 def intersection(seta,setb,result):
7   global Size
8   for i in range(Size):
9     result[i]=seta[i] and setb[i]
```

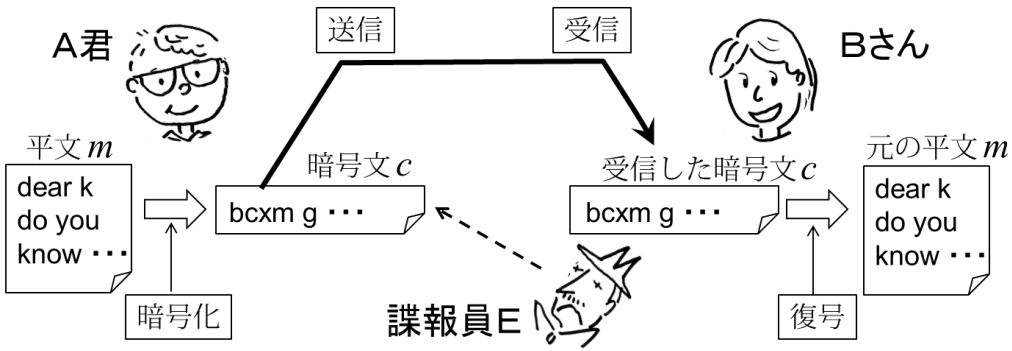
ソースコード 22: 集合演算

```
10 def complement(seta,result):
11   global Size
12   for i in range(Size):
13     result[i]= not seta[i]
14 def difference(seta,setb,result):
15   global Size
16   tmp=[-1]*Size
17   for i in range(Size):
18     complement(setb,tmp)
19     intersection(seta,tmp,result)
```

Outline

暗号通信

- 暗号システムを例に関数としての抽象化を見ていきます



暗号方式 (Cryptography)

- シーザ暗号 (Caesar chipher): ローマ皇帝シーザが使った方式
- ビジュネル暗号 (Vigenère chipher): Vigenère が作った方式
- エニグマ (enigma): 大戦中にドイツ軍が使った方式
- DES (Data Encyption Satandard), RSA (Rivest, Shamir and Adleman): 現在広く利用されている方式

シーザ暗号

- 文字を k 字先にシフトして暗号文を作る
- $k = 3$ とすると下図のようになる
- z, y, x は a, b, c になる
- これからこれを関数として表していく

a	b	c		z
↓	↓	↓	···	↓
d	e	f		c

暗号システムに必要な要素

- $\mathcal{M}$: 文の集合
- $\mathcal{C}$: 暗号文の集合
- $\mathcal{K}$: 鍵の集合
- $\mathcal{E}$: $\mathcal{M} \to \mathcal{C}$: 暗号関数の集合
- $\mathcal{D}$: $\mathcal{C} \to \mathcal{M}$: 複合関数の集合

Example (Caesar Cipher)

- アルファベットは 0 から 25 に順番に対応付けられていると仮定する
- $\mathcal{M}$: アルファベットの文字の列
- $\mathcal{C}$: アルファベットの文字の列
- $\mathcal{K}$: $\{i \mid 0 \leq i \leq 25$ であるような整数 $i\}$
- $\mathcal{E}$: $\{E_k \mid k \in \mathcal{K}$ and $\forall m(= m_1, m_2, \cdots) \in \mathcal{M}.E_k(m) = (m_i + k) \mod 26\}$
- $\mathcal{D}$: $\{D_k \mid k \in \mathcal{K}$ and $\forall c(= c_1, c_2, \cdots) \in C.D_k(k, c) = (26 + c_i - k) \mod 26\}$

いくつかの用語

- $E_k \in \mathcal{E}$ を $m \in \mathcal{M}$ に適用することを暗号化 (encipher)
- $D_k \in \mathcal{D}$ を $c \in C$ に適用することを復号 (decipher)
- $k \in \mathcal{K}$ を 鍵 (key)

平文 (plaintext)		暗号文 (ciphertext)
Hello World	暗号化 →	Hhoor Wruog
Hello World	復号 ←	Hhoor Wruog

シーザ暗号を関数であらわす

- 各文字を 0 から 25 で置き換える
  - A→0, Z→25
- 暗号化関数 $enc(3, m) = c$: 平文 $m = m_1, m_2, \cdots, m_n$，暗号文 $c = c_1, c_2, \cdots, c_n$ として $f(m_i) = (m_i + 3) \mod 26$ と表せる
- 復号関数 $dec(3, c) = m$: 平文 $m = m_1, m_2, \cdots, m_n$，暗号文 $c = c_1, c_2, \cdots, c_n$ として $f^{-1}(c_i) = ((26 + c_i) - 3) \mod 26$ と表せる

Example (Hello のシーザ暗号)

- 暗号化は 関数 enc として enc(3,"Hello")="Hhoor" と表せる
- 復号は 関数 dec として dec(3,"Hhoor")="Hello" と表せる

プログラムにしてみる
宿題 4

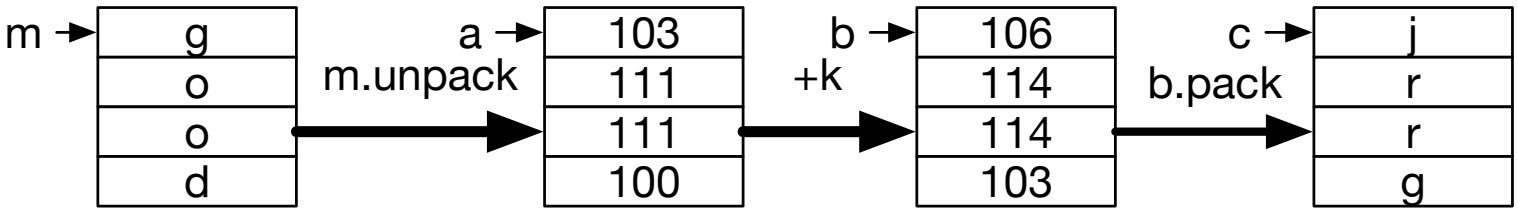
- 関係として仕様をあらわしたので，実際にプログラムにしてみます
- 具体的な計算として手続きを示した関数を定義します
- 暗号システムという複雑なものを分割

ソースコード 23: ango.py

```
1 # ango.py
2 # 暗号化サブルーチンの定義と利用
3 # 入力: 文字列
4 # 出力: 暗号化した文字列
5
6 ### Global variables
7 K = 3 # 暗号鍵の設定
8
9 # 平文を暗号化するサブルーチン
10 # enc(秘密鍵 k, 平文 m) = 暗号文 c
11 def enc(k, m):
12   ALPHABET = range(ord('a'), ord('z')+1) # 英字小文字アルファベット
13   plain = list(m.encode("ascii"))        # 文字列 -> 文字コードの配列
14   cipher = plain.copy()                  #暗号文格納用配列
15   for i,code in enumerate(plain):
16     ###
17     # 宿題 4
18     ###
19   return(bytes(cipher).decode("ascii"))
20
21 # TEST HARNESS
22 os.system("clear")
23 angobun = enc(K,input()) # 暗号文に変換
24 print(angobun)           # 暗号文を出力
```

宿題 4 (ango.py) のヒント

- ASCII Code Chart を思い出してください
- 小文字 a は 97 (0x61) が割り当てられています
- アルファベットを 0 から 25 までの数字に対応づける
  - a の文字コード 97 を引くと文字を 0 から 25 に対応付けることができる
- `m.encode("ascii")` で 1 バイトの文字コードの列に変換
- `list()` で配列を作成
- 鍵 k 分だけ各整数にたす
- 25 をこえるときは 0 にもどって計算
- `bytes(cipher).decode("ascii")` で文字の列に変換



Python における文字列の操作

- str.encode("ascii") で 1 バイトの列に変換
- list() で配列を作成
- code.py を実行すると動作が見られます

ソースコード 24: code.py

```
 1 # code.py
 2 # 文字列処理の復習用
 3 # 入力: 文字列
 4 # 出力: 文字列の文字で小文字のみ, 文字と各種情報を出力する
 5 ALPHABET = range(ord('a'), ord('z')+1)          # 英字文字アルファベット
 6 bun = input("Enter a string:")                  # 入力文字列から改行除法
 7 cc  = list(bun.encode("ascii"))                 # 文字列 → 文字コードの配列
 8 for i, moji in enumerate(bun):                  # mojiはbunのi文字目を得る (i は
       0 から始まる)
 9         code   = cc[i]                              # その文字のコードを得る
10         offset = code - ALPHABET[0]                 # 文字 a との差分
11         if code in ALPHABET:                        # 小文字アルファベットなら
12                 print(moji, ": ", code, ", ",hex(code), ", ", offset)   #   差分
                       まで表示する
13         else:                                       # そうでない時は
14                 print(moji, ": ", code, ", ",hex(code))#   差分は表示しない
```

復号関数
宿題 5

ソースコード 25: hukugo.py

```
 1 # hukugo.py
 2 # 復号サブルーチンの定義と利用
 3 # 入力: 暗号文の文字列
 4 # 出力: 復号した平文
 5 ### Global variables
 6 K = 3 # 暗号鍵の設定
 7
 8 # 平文を暗号化するサブルーチン
 9 # dec(秘密鍵 k, 暗号文 c) = 平文 m
10 def dec(k, c):
11   ALPHABET = range(ord('a'), ord('z')+1) # 英字小文字アルファベット
12   cipher = list(c.encode("ascii"))       # 文字列 -> 文字コードの配列
13   plain = cipher.copy()                   # 平文格納用配列
14   for i,code in enumerate(cipher):
15     ###
16     # 宿題 5
17     ###
18   return(bytes(plain).decode("ascii"))
19
20 ### TEST HARNESS
21 os.system("clear")
22 hirabun = dec(K,input())        # 平文に変換
23 print(hirabun)                  # 平文を出力
```

宿題 4, 5

- 宿題 4, 5 は OCW-i から
- ファイル名は暗号化プログラム: ango.py，　複合プログラム: hukugo.py としてください
- オプション:
  - 自分流の暗号方式を考えて見てください

暗号解読
課題 3 の予習

- 暗号をやぶろうとすることを暗号解読 (cryptoanalysis) と呼ぶ
- 暗号アルゴリズム ($\mathcal{E}$ と $\mathcal{D}$) は知っているが，鍵を知らないという前提
- Cipher text only attack: 暗号文だけを得ることができるとして解読 (e.g. 踊る人形)
- Known plaintext attack: 平文とその暗号文を得ることができるとして解読 (e.g. エニグマで "異常なし")
- Chosen plaintext attack: 特定の平文を送り，埋め込んだ暗号文を得ることができるとして解読 (e.g. 日本軍のミッドウェイ島攻撃)

暗号解読のヒント

- Cipher text only attack で解くこと
- 統計的な手法を用いる
  - 文字の出現頻度: 踊る人形，コナン・ドイル (ホームズのはなしのことかな)

Part VII

CS 第 1—課題 3

CS 第 1 —課題 3

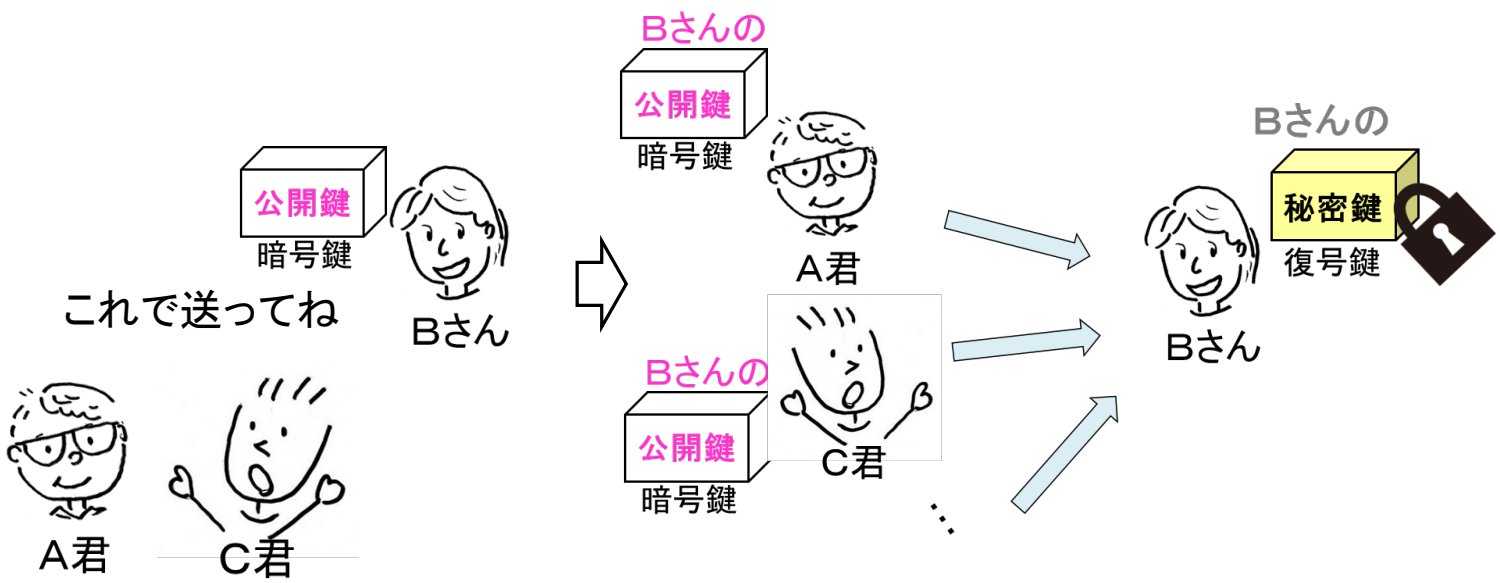
Outline

現代の暗号方式
Optional

- シーザ暗号では暗号化と復号に同じ鍵をもちいていました
- 暗号化と復号で異なる鍵を使用する暗号について紹介します
  - 公開鍵暗号方式 (Public Key Crpptography)

公開鍵暗号方式

- Bob さんは Alice さんに自身の公開鍵を送る
- Bob さんにメッセージを送るときは Alice さんは Bob さんの公開鍵で暗号化する
- Alice さんはメッセージを受け取ったら，自身の秘密鍵で復号する
- 逆に，Alice さんが送るときは Alice さんの秘密鍵で暗号化し， Bob さんは Alice さんの公開鍵で復号する



公開鍵暗号方式 (Public Key Cryptography)

- Diffie-Hellman 方式
- RSA

Public key cryptosystem の特徴

1. 適当な鍵が与えられているときはメッセージを暗号化, 復号することが容易
2. 公開鍵から秘密鍵を予測することは困難

Diffie-Hellman Key Exchange

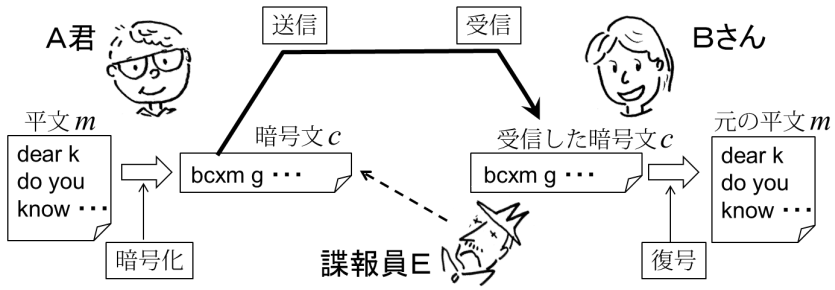
原理

- $b^k = g \mod p$ の k を求める問題に置き換えられる

1. 大きな素数 p と g の組をひとつ選ぶ (RFC 3526)
   - $p = 17, g = 3$
2. Alice と Bob はそれぞれ秘密鍵をえらぶ
   - $k_{private}^{Alice} = 15, k_{private}^{Bob} = 13$
3. 公開鍵をそれぞれ計算して交換する
   - $k_{public}^{Alice} = g^{k_{private}^{Alice}} \mod 17 = 3^{15} \mod 17 = 6$
   - $k_{public}^{Bob} = g^{k_{private}^{Bob}} \mod 17 = 3^{13} \mod 17 = 12$
4. Bob は Alice の公開鍵で共有秘密鍵を計算する
   - $S_{Bob,Alice} = (k_{public}^{Alice})^{k_{private}^{Bob}} \mod p = 6^{13} \mod 17 = 10$
5. Alice は Bob の公開鍵で共有秘密鍵を計算する
   - $S_{Alice,Bob} = (k_{public}^{Bob})^{k_{private}^{Alice}} \mod p = 12^{15} \mod 17 = 10$

課題 3 のシチュエーション

- Alice と Bob が通信しているとして，そこに Eve (eavesdropper) がいるとします
- 通信の内容を盗み見られないように暗号化して通信します
- 暗号化の手順は以下の通り
  1. 送りたい文 (平文 m) を暗号化 (encryption) して暗号文 c を作成
  2. 暗号文 c を送信
  3. 暗号文 c を受信
  4. 暗号文 c を復号 (decryption) して平文 m を得る
- Eve はこの通信の内容を盗み見ようと試みる (暗号解読を試みる)



課題 3 ガイド

課題 3 テーマ

暗号解読に挑戦

- 課題 3 で作成してほしいプログラム
  - 暗号解読プログラム: kaidoku.py
  - オプション:
    - 講義で説明していない暗号方式 myango.py, myhukogo.py と
    - その暗号を解読するための自分流のプログラム mykaidoku.py

作業内容

- code.py での文字列操作を参考に ango.py, hukugo.py を完成させてみてください
- それが終わったら kaidoku.py を作ってみてください
- リダイレクトやパイプを使う

暗号化するとき

```
python3 ango.py < plaintext.txt
```

復号するとき

```
python3 hukugou.py < chiphertext.txt
```

暗号化して復号するとき

```
python3 ango.py < plaintext.txt | python3 hukugo.py
```

リダイレクションとパイプ

- OS には標準入力や標準出力という概念があります
- 通常は標準入力はキーボード，標準出力はディスプレイを指しています
- プログラムが入力を読み取る場合，特に指定がなければ標準入力から読み込み，
- 出力する場合は標準出力に書き込みます
- リダイレクションは標準入出力をファイルに変更します
- パイプ | はプログラムの出力を別のプログラムの入力につなげる役割します
- >! とびっくりをつけると上書きします

入力リダイレクション

```
python3 ango.py < plaintext.txt
```

出力リダイレクション

```
python3 ango.py > chiphertext.txt
```

パイプ

```
python3 ango.py | python3 hukugo.py
```

暗号解読のヒント

- 意味をなす一般的な文章では文字の出現頻度には偏りがあります
- たとえば英語では母音 e が最も出現頻度が高い
- シーザ暗号はこの出現頻度は暗号化しても偏りは変わりません
- この特徴を利用して何文字移動しているかを予測することができます
- 各文字 26 個の頻度を計算するために出現回数を要素とする配列をつくる

英語の場合

一番多く現れる文字が e のはず！

qxuvnb qjm knnw bnjcnm oxa bxvn qxdab rw brunwln frcq qrb
uxwp, cqrw kjlt ldaenm xena j lqnvrlju enbbnu rw fqrlq qn
fjb kanfrwp j yjacrldujauh vjuxmxaxdb yaxmdlc. qrb qnjm
fjb bdwt dyxw qrb kanjbc, jwm qn uxxtnm oaxv vh yxrwc xo ...

n が19回出現で最多

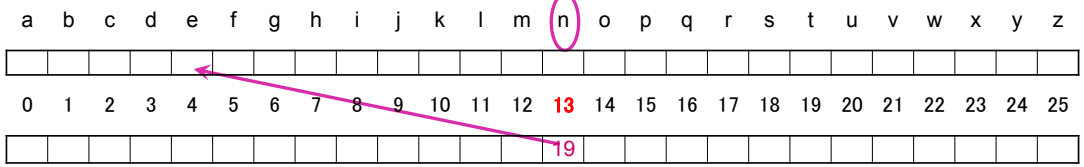
暗号解読のヒント—つづき

- 13 番目の文字 n が最多なので 4 番目の文字 e にシフト
- 13 – 4 = 9 なので 9 文字シフトしていると推測できる

qxuvnb qjm knnw bnjcnm oxa bxvn qxdab rw brunwln frcq qrb
uxwp, cqrw kjlt ldaenm xena j lqnvrlju enbbnu rw fqrlq qn
fjb kanfrwp j yjacrldujauh vjuxmxaxdb yaxmdlc. qrb qnjm
fjb bdwt dyxw qrb kanjbc, jwm qn uxxtnm oaxv vh yxrwc xo ...

差分

n が19回出現で最多



暗号解読のヒント—より高度な方法

- フォーマルには各文字の出現頻度と暗号文での出現頻度の相関 $\phi(i)$ をとります
- $\phi(i) = \Sigma_{0 \leq c \leq 25} f(c)p(c-i)$, ここで $f(c) = \frac{n_c}{l_{ct}}$ は暗号文での文字 c の出現頻度, $p(c-i)$ は一般の出現頻度とする
- https://sites.google.com/a/presystems.xyz/sample/home/elementary-computer-science に置いてある 1-gram.txt が出現頻度のファイルです
- 相関係数 $\phi(i)$ が
  - 1 に近いほど: $f(c)$ が大きくなれば $p(c-i)$ も大きくなり相関が強くなる
  - 0 近傍: $f(c)$ と $p(c-i)$ はあまり相関がない