コンピュータサイエンス入門第一 —3a,4b(CS1)—

永藤 直行

東京工業大学

3rd quarter

Part I

Prologue

Prologue

1. ガイダンス
   - 科目の概要
   - 参考図書
   - 評価基準
2. CS 講義概要
   - 講義の目標
   - 演習内容
3. 情報の格納
   - 教育用計算機ステムの利用開始
   - ファイル
   - ディレクトリ (フォルダ)
   - Terminal について
   - ソースファイルの編集

Outline

1 ガイダンス
- 科目の概要
- 参考図書
- 評価基準

2 CS 講義概要
- 講義の目標
- 演習内容

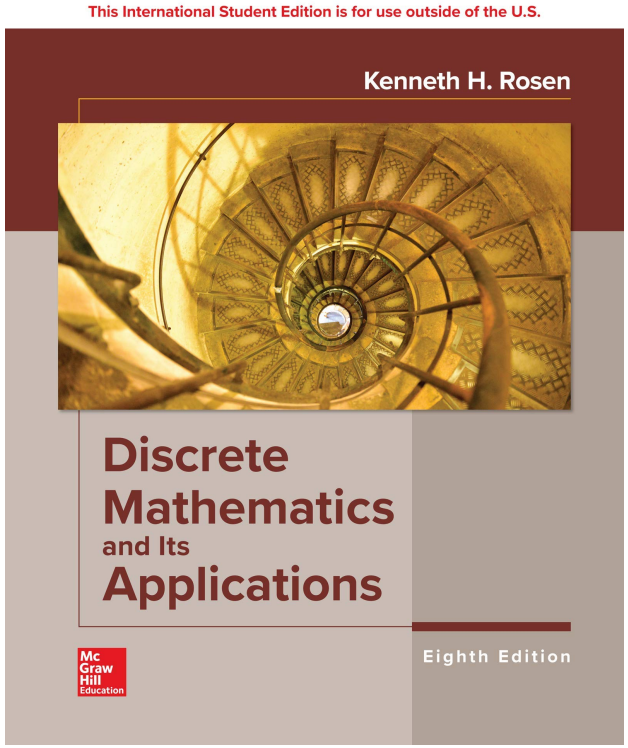
3 情報の格納
- 教育用計算機ステムの利用開始
- ファイル
- ディレクトリ (フォルダ)
- Terminal について
- ソースファイルの編集

科目の概要

- 学期: 水曜日 1-2 限と水曜日 3-4 限
- 場所: 南 4 号館 3 階第 2 演習室
- 担当教員: 永藤 直行 (ナガトウ ナオユキ)
- 問い合わせ先: nagatou@presystems.xyz
- 質問はメイルで受け付けます．必要な場合には Zoom で
- 永藤担当クラスのサイト:
  https://sites.google.com/presystems.xyz/elementarycs/top
- 共通サイト: https://wakita.github.io/classes/years/y22/cs1/course.html
- その他ツールは必要に応じて URL を示します

参考図書

- 渡辺治 著, コンピュータサイエンス–計算を通して世界を観る, 丸善出版 (2015)
- Kenneth H. Rosen 著, Discrete Mathematics and Its Applications 8th ed.(2018)



評価基準

- 講義は全 7 回，期末試験は行いません
- 宿題: 3 回 くらい（提出不要）
- 課題: 3 回 25 + 30 + 30 = 85 点（必須）
- 特別課題: 1 回 15 点（提出任意）
- 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

Outline

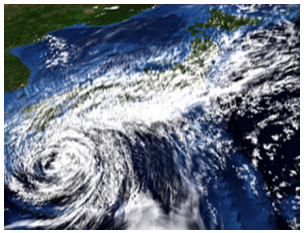
1 ガイダンス
- 科目の概要
- 参考図書
- 評価基準

2 CS 講義概要
- 講義の目標
- 演習内容

3 情報の格納
- 教育用計算機ステムの利用開始
- ファイル
- ディレクトリ (フォルダ)
- Terminal について
- ソースファイルの編集

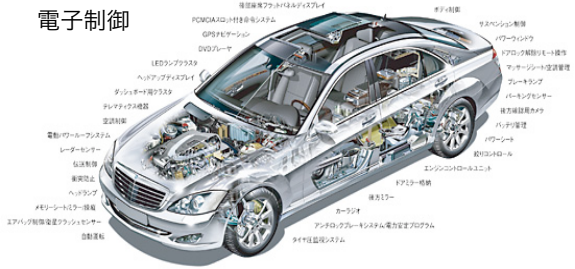
講義の目標

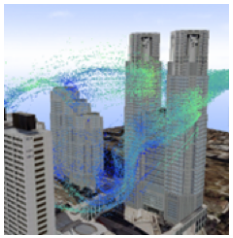
- 本講義では，このコンピュータサイエンスの基本をなす考え方を，課題を通して体得する
- 物理現象をシミュレートしたり
- 経済活動にともなう帳票類を管理したり
- 機器を制御したり
- コンピュータがいろいろな場面で利用されている



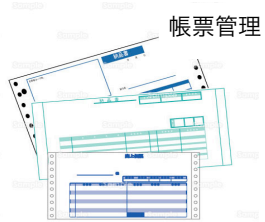
シミュレーション

電子制御





帳票管理



情報クラウド

なぜコンピュータが利用できるのか

コンピュータに載せるとは

対象を計算をもちいて表現し，コンピュータに処理させる

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

演習内容

- 以下の演習を予定しています

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
  - 文字列
  - サブルーチン
- 課題 S: 自然数から整数への拡張

Outline

1 ガイダンス
- 科目の概要
- 参考図書
- 評価基準

2 CS 講義概要
- 講義の目標
- 演習内容

3 情報の格納
- 教育用計算機ステムの利用開始
- ファイル
- ディレクトリ (フォルダ)
- Terminal について
- ソースファイルの編集

教育用システムの利用

- 教育用電子計算機システムのサイトの “アカウント利用開始” を参照

情報の格納

ファイルとディレクトリ (フォルダ)

- さまざまなタイプの情報をビットの列として記録しいてます
  - 数値，文字，図，表，写真，音声，動画など情報のあらゆるタイプ
- コンピュータの中では情報をファイルやディレクトリ (フォルダ) と云うもので管理しています
- ファイル
  - コンピュータ内の情報はファイルという単位で扱われる
  - e.g. テキストファイル，画像ファイル，音声ファイルなど
- ディレクトリ
  - ファイルの入れ物
- 名前はすきなものをつけられます
  - でも漢字は使わないほうがいいです
  - アルファベットと記号だけで名前を付けるほうがいいです

ファイルの形式

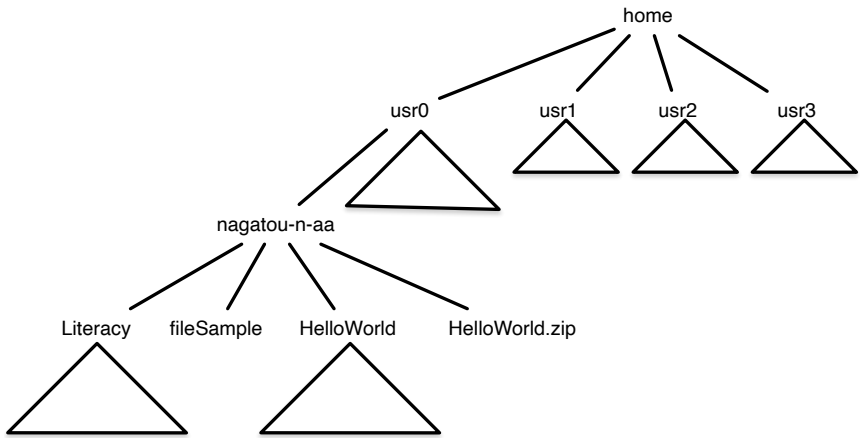
- さまざまな情報がビット列であらわされ，コンピュータで編集，加工することができます
- 編集，加工するにはプログラムが必要になります
- しかし，ビットの列というだけでは何のデータか分かりません
- ファイルに納められているデータが何のデータであるか約束が必要です
- それがファイル形式ということになります
- ファイルの形式は拡張子 (suffix) と呼ばれるもので示されていることが多い
  - .csv, .txt, .c, .obj, .pdf, .doc, .mid などなど
- 形式ごとにプログラムが対応づけられています

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

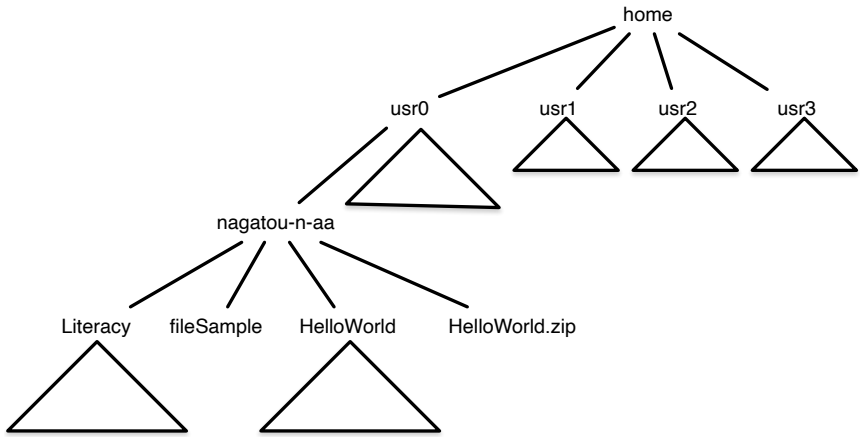
ディレクトリの階層

- ファイルはすべて異なる名前をつけなければなりません
- ファイルはだんだん増えて違う名前を考えるのは難しくなるかもしれません
- 複数のユーザが利用しているので知らないうちに同じなまえになっているかもしれません
- 階層化して管理すると便利です
- 特定のディレクトリ内の高々数個のファイルならば違う名前を付けるのは容易なはず



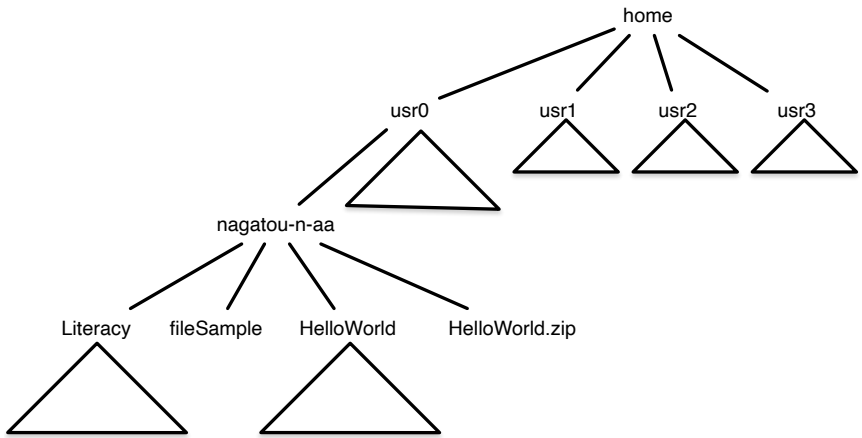
作業ディレクトリあるいは current directory

- ファイルをディレクトリごとに整理できたら，操作は特定のディレクトリのファイルを対象にするとおもいます
- 作業するときには利用者がその場所まで移動します
  - 場所とはいってもコンピュータ内でのこと
- 自分が今いるディレクトリを作業ディレクトリあるいは current directory と呼びます



パス

- コンピュータ内の場所はパス (path) で表します
- パス (path) は /でディレクトリ名を区切った文字列
  - e.g. `~/literacy` と云ったような文字列
- ディレクトリにはルートディレクトリとホームディレクトリと呼ぶ特別なディレクトリがあります
- 絶対パスと相対パス



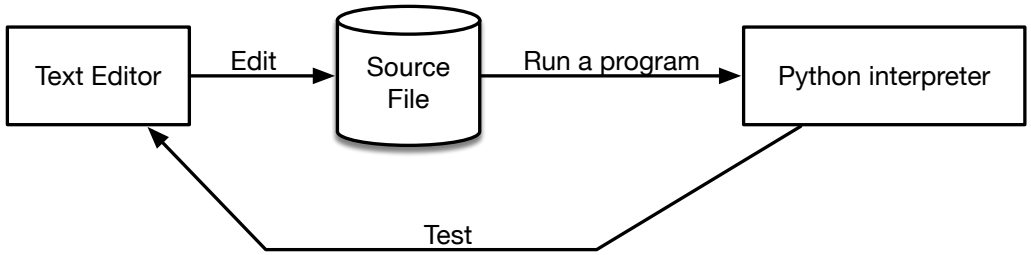
ディレクトリに対する操作

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

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

プログラム開発の流れ



ターミナルを使ってみよう

- ターミナルを使ってプログラムを実行してみよう
- 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 (例えば CS1) を作成
- https://sites.google.com/presystems.xyz/elementarycs/top から必要なファイル (gcd.py) を作成したディレクトリにダウンロード
  - ホームディレクトリの下の Downloads に保存されるかも

準備

```
> mkdir CS1          # 課題用のディレクトリを作成
> cd CS1             # 課題用ディレクトリに移動
> python3 gcd.py     # プログラムを実行
```

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 Emacs gcd.py
```

あるいは

```
> open -a Macvim gcd.py
```

Part II

計算の基本

計算の基本

4 CS のこころ

5 コンピュータの中では

- Bit と Byte
- 自然数の n 進表記
- コンピュータの中での計算

6 計算は ±1 と繰り返し

- 計算とは
- 計算の基本要素
- Python プログラムの書き方
- 宿題 1 を動かしてみる

Outline

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（数）として表すこと
  - 処理を基本演算の組み合わせで表すこと
- 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
- このプログラムを実行することでコンピュータ上で再現させる

Outline

データは数

- データはすべてビットの列として表される
  - 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，事象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

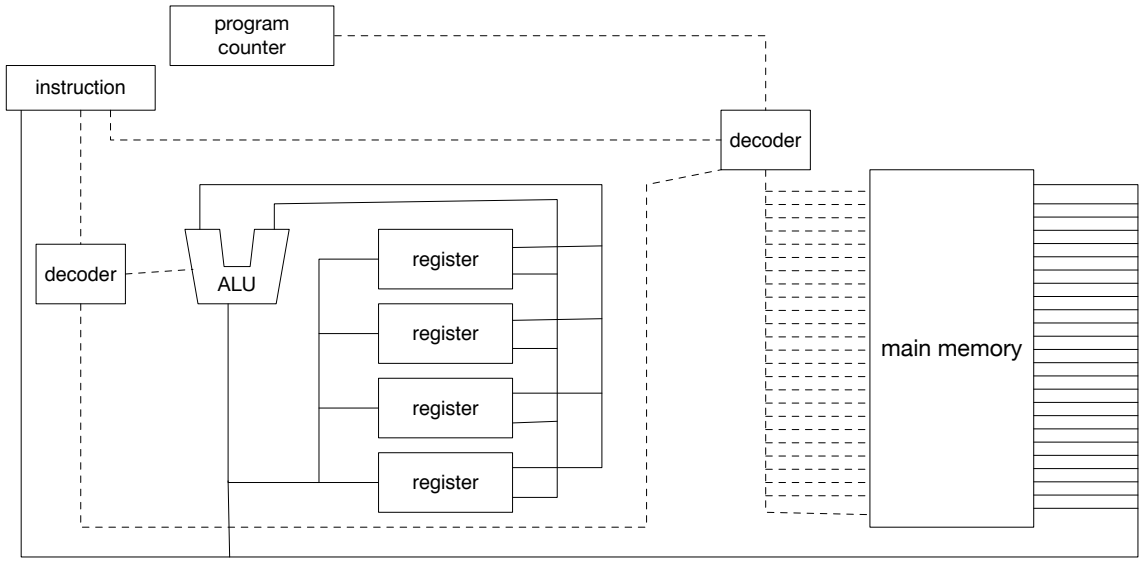
- m 進数から n 進数への変換
- 手始めに 10 進数から 2 進数への変換

Example (10 進 ⇔2 進)

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

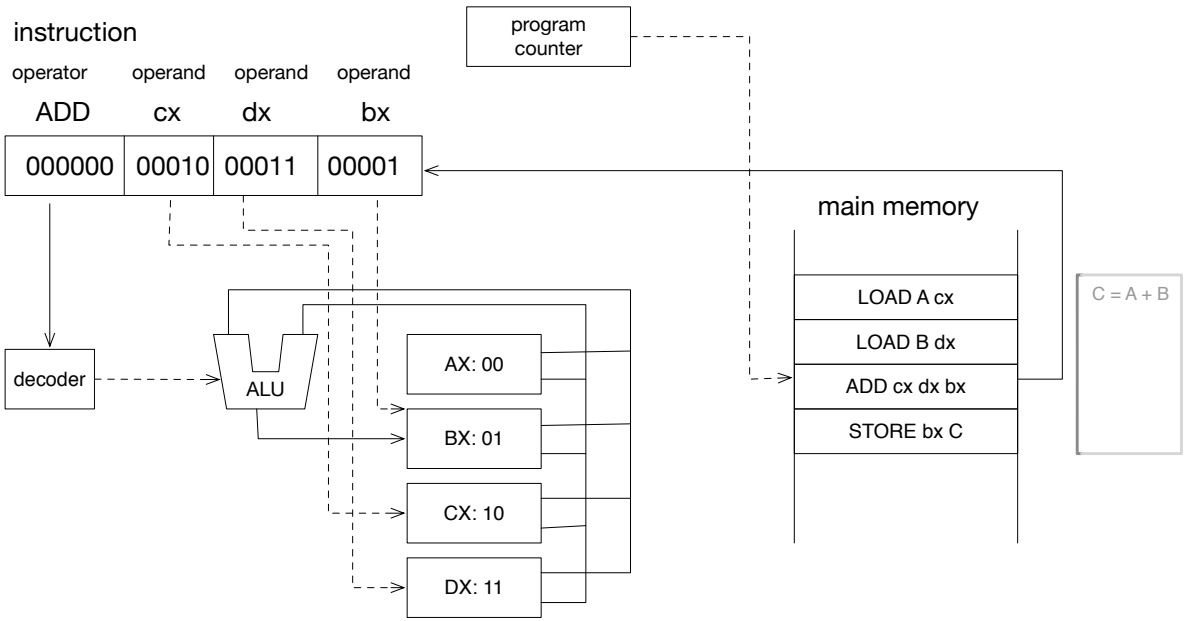
コンピュータの中では

- コンピュータの命令自体も符号化されてます
- CPU (Central Processing Unit) ごとに命令セットも符号も異なっています
- ここでは CPU が命令を実行するサイクルについて見てみます



演算のサイクル

1. instruction に命令をフェッチ
2. メインメモリからレジスタにデータを移動
3. ALU (Arithmetic and Logic Unit) がレジスタからデータを取り出す
4. ALU で演算
5. 結果をレジスタに書き込む
6. レジスタからメインメモリにデータを移動
7. ADD cx dx bx という命令を例にすると下図のようになります



Outline

計算とは

- これまで自然数から初めて実数はては虚数のうえでの演算や関数の規則を覚えていろいろな計算をしてきた
- 挙句には証明するといったこともしてきた
- これらに共通する特徴を捉えて計算というものを体系的に捉えていく
- 計算には入力と出力があって入力から出力を生成する
- 生成過程を定めたものがアルゴリズム (algorithm)

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

Listing 1: gcd.py

```
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

- では，アルゴリズム的に計算可能なものとは何か？
  - Hilbert's Program
  - 数学基礎論，Computability といったキーワード
- 有限の操作を加算個並べて表せるものをアルゴリズム的に計算可能ということにする
- アルゴリズムを記述するための最も基本となるものは何か？

基本要素

- ある値に ±1 する操作
- 値を保持したり読み出す操作
- 条件付きジャンプ
  - 繰り返し
  - 条件分岐

基本要素だけの四則演算

- $m + n$ は m に +1 を n 回施す
- $m \times n$ は m に +1 を m 回施すことを n 回施す

Python における基本式

- 式の基本要素は 整数，実数がある
  - 整数の例: 286, 386, 486
  - 講義では自然数だけ扱います
- 基本要素と演算子 +, −, *, //, %, ** などがある
- 数と演算子を組み合わせて式を作ることができる
  - 入れ子にできます: 286 + (386 + 486)
  - 今回は +1 と −1 だけ
- 式には名前をつけ（この名前のことを変数と呼ぶ），変数に式の値を保持し，変数で値を参照することができます
  - 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

- 式を上から下へ順番に並べることで合成
- 上から下へ順番に実行される
- 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

繰り返し文

while 文

- 特定の実行列を条件式が真である間繰り返し実行
- 詳細は Python ドキュメント 8.2, 8.3 節参照

Listing 2: while 文

```
1 while 条件式 :
2   文または式
3   文または式
```

論理演算子

算術演算子	使用例	意味
+	x + y	x と y の足し算
-	x − y	x と y の引き算
*	x * y	x と y のかけ算
//	x // y	x を y 割った商
%	x % y	x を y 割ったあまり
**	x ** y	x の y 乗
論理演算子	**使用例**	**意味**
==	x == y	x と y が等しいなら真
!=	x != y	x と y が等しくないなら真
>=	x >= y	x は y 以上なら真
<=	x <= y	x は y 以下なら真
>	x > y	x は y より大きいなら真
<	x < y	x は y より小さいなら真

論理の合成

- 条件式は Python ドキュメント 6.10, 6.11, 6.12 節参照

論理演算子	使用例	意味
and	*p* and *q*	論理積: 両方が真なら真
or	*p* or *q*	論理和: いずれかが真なら真
not	not *p*	*p* の否定

基本要素だけの加算

Listing 3: add.py

```
 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

Listing 4: mult_basiconly.py

```
 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
```

条件分岐

if 文

- 条件式が真の場合だけ文や式を実行
- else がある場合は条件式が偽のとき実行
- 詳細は Python ドキュメント 8.1 節参照
- 条件式は Python ドキュメント 6.10, 6.11, 6.12 節参照

Listing 5: gcd.py(if 文と for 文の例)

```
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))
```

宿題 1 を動かしてみる

宿題 1 –±1 だけで四則演算を作る

- https://sites.google.com/presystems.xyz/elementarycs/top に div-skeleton.py と sub-skeleton.py があるはずなのでそれを完成させて実行する

Listing 6: 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)
```

Listing 7: div.py

```
1  # div.py
2  # Input: 自然数 x, y
3  # Output: x æ 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)
```

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計算の仕組みの説明をソースコードに埋め込む
- Python 言語の説明は不要
- 提出は T2Schola から

ひつじさんを動かしてみる

- 大きな数字を複数定義して 1 枚の紙に見立てる
- 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
- ここでは 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

Listing 8: sheep.py (declaration)

```
 1 ################
 2 # pattern def. #
 3 ################
 4 d0  = 10000000000000000000000000000000000000000000000000000000000000
 5 d1  = 10000000000000000000000011111111000000000000000000000000000000
 6 d2  = 10000000000000000000000111000000011000000000000000000000000000
 7 d3  = 10000000111111111111110011000011000111111000000000000000000000
 8 d4  = 10000011100000000000000011001110001100001100000000000000000000
 9 d5  = 10000011000000000000000011111001100010001100000000000000000000
10 d6  = 10000011000000000000000000000000000000111100000000000000000000
11 d7  = 10000011000000000000000000000000000001110000000000000000000000
12 d8  = 10000011000000000000000000000000000011100000000000000000000000
13 d9  = 10000011000000000000000000000000000110000000000000000000000000
14 d10 = 10000011000000000000000000000000001110000000000000000000000000
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
- d0 を足しているのは絵がずれないようにするため

Listing 9: sheep.py (shift)

```
18        # shift #
19        d1  = (d1 -  d0) // 10 + d0
20        d2  = (d2  - d0) // 10 + d0
21        d3  = (d3  - d0) // 10 + d0
22        d4  = (d4  - d0) // 10 + d0
23        d5  = (d5  - d0) // 10 + d0
24        d6  = (d6  - d0) // 10 + d0
25        d7  = (d7  - d0) // 10 + d0
26        d8  = (d8  - d0) // 10 + d0
27        d9  = (d9  - d0) // 10 + d0
28        d10 = (d10 - d0) // 10 + d0
29        d11 = (d11 - d0) // 10 + d0
30        d12 = (d12 - d0) // 10 + d0
31        d13 = (d13 - d0) // 10 + d0
```

Part IV

配列

配列

Outline

配列

- いくつかのデータオブジェクトを 1 つにまとめて扱いたい時がある
- 配列は，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

Listing 10: 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 =  10000000001111111100000000000000
9 d9 =  10000000000000000000000000000000
```

Listing 11: 配列

```
1  d = [10000000000000000000000000000000,
2       10000000001100001100000000000000,
3       10000000001100001100000000000000,
4       10000000000000000000000000000000,
5       10000011000000000000011000000000,
6       10000001100000000000110000000000,
7       10000000110000000001100000000000,
8       10000000001111111100000000000000,
9       10000000000000000000000000000000]
```

配列の参照と代入

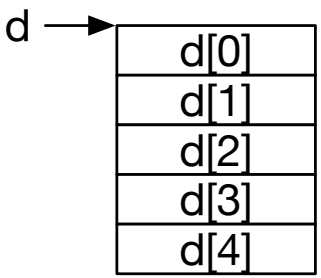
- 連続したメモリ領域を確保
- 先頭を 0 としてインデックスで参照 (ここは板書します)

Listing 12: 代入

```
1  d = [0]*9
2  d[0]=100000000000000000000000000000000
3  d[1]=100000000011000011000000000000000
4  d[2]=100000000011000011000000000000000
5  d[3]=100000000000000000000000000000000
6  d[4]=100000110000000000000011000000000
7  d[5]=100000011000000000000110000000000
8  d[6]=100000001100000000011000000000000
9  d[7]=100000000011111111000000000000000
10 d[8]=100000000000000000000000000000000
```

Listing 13: 参照

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



配列の例

- 総和を求める
- Listing 14: 整数の配列を宣言する例
- Listing 15: 入力した整数を配列にする例

Listing 14: sum6.py

```
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)
```

Listing 15: sum.py

```
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/presystems.xyz/elementarycs/top 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

Listing 16: 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

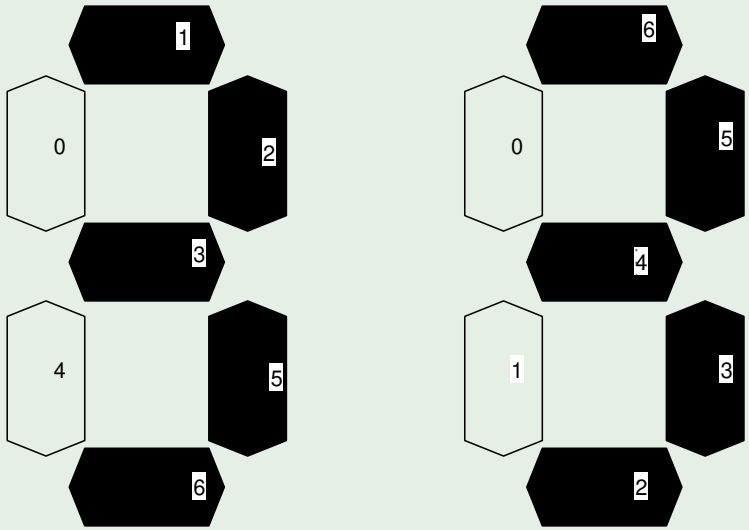
文字コード

- コードとは，文字や記号をコンピュータで扱うための符号です
- 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

Listing 17: 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 import os
 6
 7 os.system('clear')
 8 str = (input("strings? ")).encode("ascii")
 9 print(str)
10 for k in range(0,len(str)):
11    print(chr(str[k]), hex(str[k]))
```

出力例

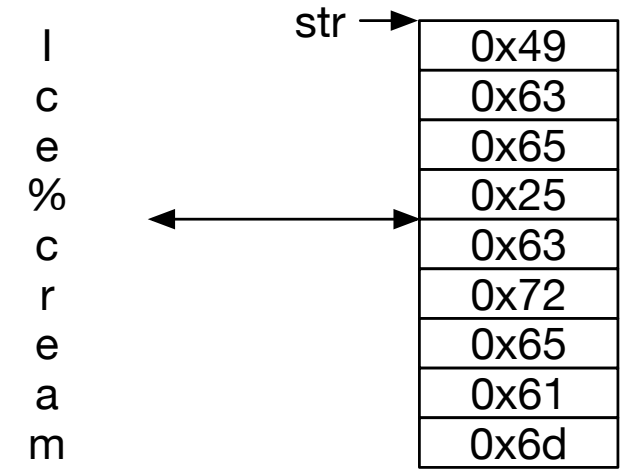
```
> python3 stringPrint.py
strings? Ice%cream
I 0x49
c 0x63
e 0x65
% 0x25
c 0x63
r 0x72
e 0x65
a 0x61
m 0x6d
```

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/presystems.xyz/elementarycs/top の abcPrint-skeleton.py を完成させる

Listing 18: 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
```

Outline

課題 S

- 自然数上の演算を整数に拡張
- https://sites.google.com/presystems.xyz/elementarycs/top からまずは integers-skeleton.py をダウンロード
- お話していない部分も含まれるので調べながらやってみてください
- ソースコード中のコメントを参照して (1)-(4) が虫食いになっているので完成させる
- 提出は出来たところまででいいです
- 出来なかったところはコメントにしてください

Part V

CS 第 1—課題 2

CS 第 1 —課題 2

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，自然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 https://sites.google.com/presystems.xyz/elementarycs/top から junkan.py をダウンロードして実行してみてください
- たとえば, 3 と入力すると停止しないと思います.
- 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

Listing 19: junkan.py

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

Part VI

関数とサブルーチン

関数とサブルーチン

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
  - 配列: データの集まりに名前をつけて抽象化
- 今回は計算の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演算子のように抽象化

Python における関数定義

- Python では def というキーワードをつかって関数を定義します
- 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

関数定義の例

Listing 20: mult_basicsonly.py

```
1  seki = 0          # seki を 0 で初期化
2  while y > 0:      # y が 0 より大きい間繰り返す
3    a = seki
4    b = x
5    wa = a          # 和のプログラム add.py を挿入
6    while b > 0:
7      wa = wa + 1
8      b = b - 1
9    seki = wa       # wa の値 (seki + x) を seki に代入
10   y = y - 1       # y - 1 の値を y に代入
```

Listing 21: mult_basicsonly.py

```
11 def add(x,y):
12   wa = x
13   while y > 0:
14     wa = wa + 1
15     y = y - 1
16   return(wa)
17 def mult(x,y):
18   seki = 0
19   while y > 0:
20     b = x
21     seki = add(seki,b)
22     y = y - 1
23   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

Listing 22: bit_string.py

```
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]
```

Listing 23: bit_string.py

```
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)
```

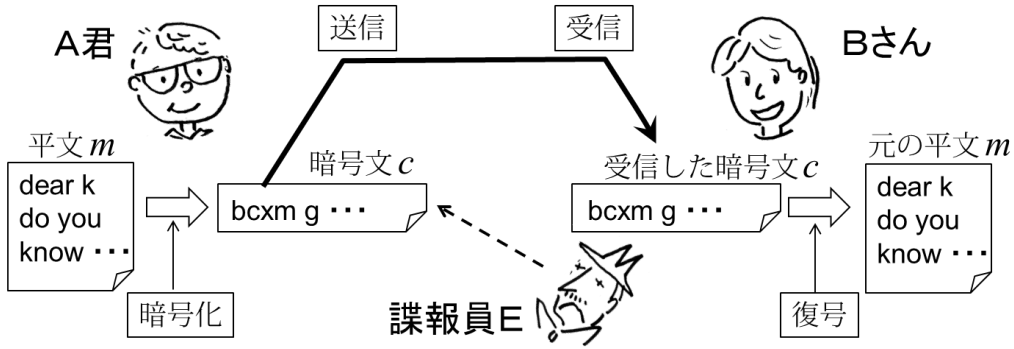
仮引数と実引数

- 関数は仮引数というものをもつ
- 複数の関数が同じ名前の仮引数を持っていても良い
- 仮引数は関数の本体で有効である
- 関数を呼び出したときの値に束縛 (bind) されて，関数の本体では呼び出し時の値に置き換えられる
- 呼び出し時の値を実引数という
- 一般に変数は有効範囲 (scope) が決まっている
- 仮引数は関数本体が有効範囲である

Outline

暗号通信

- 前回まで，合成した計算の抽象化として関数を見てきました
- 今回は，現実世界の対象を関数として表すというのを見ていきます
- 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 cipher): ローマ皇帝シーザが使った方式
- ビジュネル暗号 (Vigenère cipher): 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
- $\mathcal{D}$: $\mathcal{C} \to \mathcal{M}$: 復号関数の集合

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 $\forall c(= c_1, c_2, \cdots) \in \mathcal{C}.D_k(k, c) = (26 + c_i - k) \mod 26\}$

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

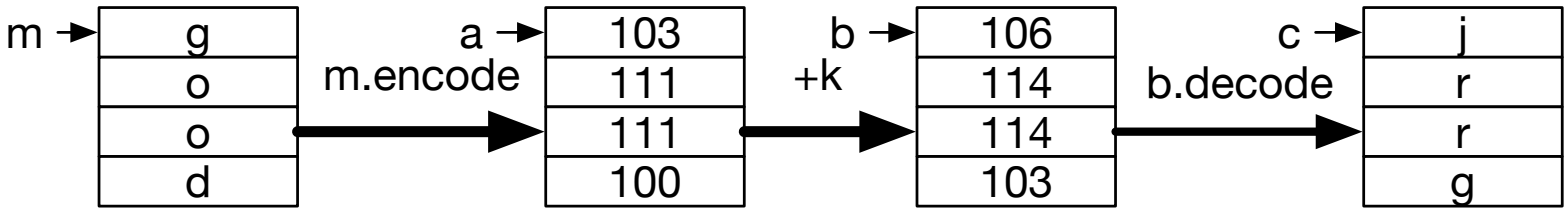
- 関係として仕様をあらわしたので，実際にプログラムにしてみます
- 具体的な計算として手続きを示した関数を定義します
- 暗号システムという複雑なものを分割

Listing 24: 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"))
```

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

Listing 25: 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

Listing 26: 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"))
```

宿題 4, 5

- https://sites.google.com/presystems.xyz/elementarycs/top から ango-skeleton.py と hukugo-skeleton.py をダウンロード
- 同様にテスト用サンプルデータ plaintext.txt と ciphertext.txt をダウンロード（ソースコードと同じディレクトリに保存してください）

Part VII

CS 第 1—課題 3

CS 第 1 —課題 3

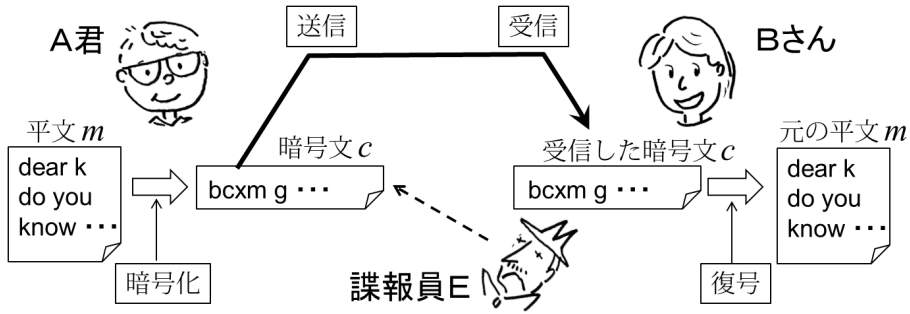
Outline

暗号解読

- 暗号をやぶろうとすることを暗号解読 (cryptoanalysis) と呼ぶ
- ここでは暗号アルゴリズム ($\mathcal{E}$ 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)

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て暗号文だけが入手可能で暗号解読を試みる



課題 3

- 暗号解読に挑戦
- https://sites.google.com/presystems.xyz/elementarycs/top に置いてある kaidoku-skeleton.py を参考に暗号解読プログラムを kaidoku.py を作成してください
- 課題 3 で作成してほしいプログラム
- 提出は T2Schola からソースコードを提出

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！

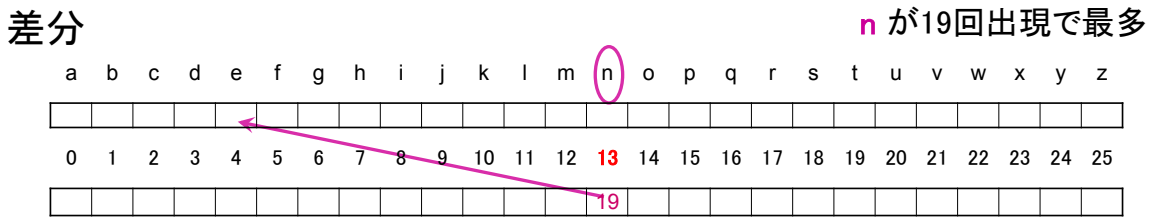
qxuvnb qjm knnw bnjcnm oxa bxvn qxdab rw brunwln frcq qrb
uxwp, cqrw kjlt ldaenm xena j lqnvrlju enbbnu rw fqrlq qn
fjb kanfrwp j yjacrldujauh vjuxmxaxdb yaxmdlc. qrb qnjm
fjb bdwt dyxw qrb kanjbc, jwm qn uxxtnm oaxv vh yxrwc xo ...

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 ...



暗号解読のヒント—より高度な方法

- より高度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/presystems.xyz/elementarycs/top に置いてある 1-gram.txt が出現頻度のファイルです
- $\phi(i)$ 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