会津大学
THE UNIVERSITY OF AIZU

大学等の「復興知」を活用した人材育成基盤構築事業
2023 年度若手人材が輝くロボット・ICT 人材育成プログラム

Aizu Online Judge コース問題解説

Python 情報1・プログラミング

The University of Aizu.

目次

はじめに

このテキストは、Aizu Online Judge のコース「情報1（INFO1）」の問題を Python によるプログラミングによって解くための基本文法を学びます。ステップ by ステップで演習問題を解いていく実践的なプログラミング演習です。このコースを修了すれば、Python の基本的な文法を理解し、Python によって、データの探索や整列などの基礎的なアルゴリズム(問題を解く手法)が記述できるようになります。さらに、より実用的なアプリケーションを開発するための学習の準備が整います。

演習問題は、プログラミングの入門者・初心者を対象としており、各トピックで初歩的な問題を解いていきます。トピックの最初に設けられたトピック解説と、各問題に必要に応じて準備された問題解説を参考にしながら、出題された問題を確実に解いていくことができます。

演習問題は全て Aizu Online Judge（AOJ）に収録されており、採点をしながら演習問題を解き進めていくことができます。また、学習進捗を確認し楽しみながら継続的に学習を進めることができます。AOJ には、以下の URL からアクセスしてください。

https://onlinejudge.u-aizu.ac.jp/beta/ice

まずは、ヘルプ→ツアーでシステムの使い方を確認してください。より詳しい使い方は、ヘルプ→チュートリアルから確認できます。

本テキストの演習問題については、問題集→コース→情報(INFO1)からトピック演習を開くことができます。

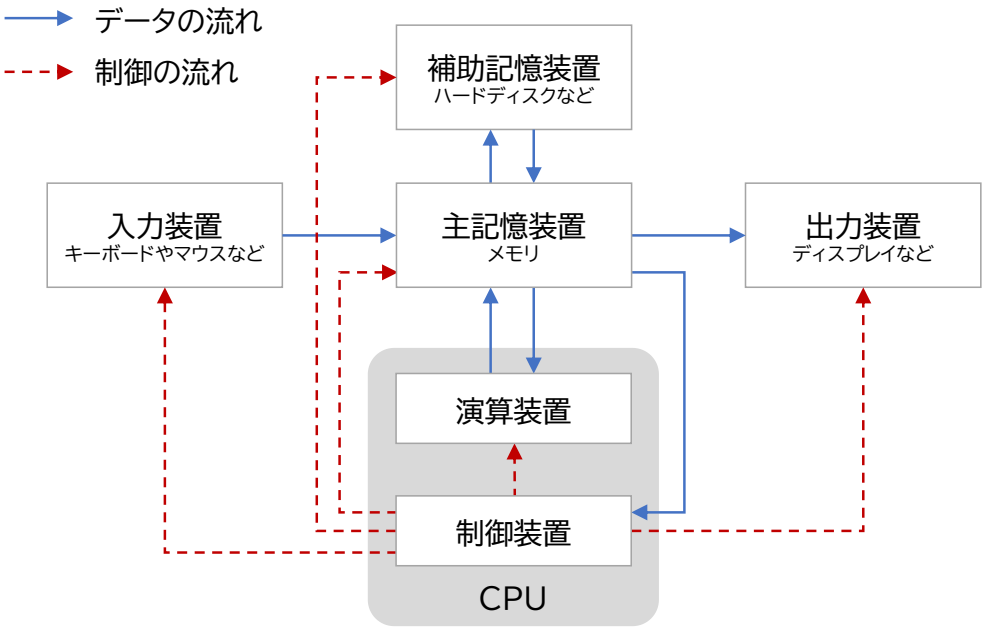
一方、AOJ には、様々なコースと豊富なチャレンジ問題が収録されています。チャレンジ問題は、画面上部の「問題 ID を入力」から開くことができます。本講座を修了した後でも、より発展的なコースとチャレンジ問題を活用し、継続的にプログラミングスキルを磨くことができます。

コース解説

コース INFO1 では、プログラミングの入門者を対象に、プログラミング言語の基本的な文法を身に付けます。具体的には、各プログラミング言語における、入出力処理、変数(メモリ)の扱い、処理の分岐と繰り返し、配列(連続するデータの要素)の利用に関するコーディング方法を学びます。このコースを修了すれば、現在学んでいるプログラミング言語によって、データの探索や整列などの基礎的なアルゴリズム(問題を解く手法)が記述できるようになります。

プログラムとは

プログラミングとは、プログラムを作成する知的な作業です。プログラムとは、コンピュータに実行させる命令文の集まりです。そこで、コンピュータの装置を見ながら「プログラムとは何か？」を考えてみましょう。



上の図は、コンピュータの5大装置、それらの間のデータの流れと制御の流れを表します。コンピュータの中核は、その頭脳となる CPU とメモリです。メモリに計算結果を記憶しながら、CPU で計算を行います。プログラム自体は、メモリに取り込まれ、その内容が制御装置によって実行されます。

プログラムが行う「処理」に着目すると、プログラムとは、「入力」されたデータに基づいて、「記憶」、「演算」、「制御」を行いながら、目的の「出力」データを得るための仕組みであることが分かります。このテキストでは以下の通りに、これらのキーワードに関するトピックでプログラミングを学びます。

出力

コンピュータがデータを出力する先は、ディスプレイ、ファイルシステム、プリンタ、ネットワークなど様々です。その中でも、最も基本的な出力先が標準出力です。標準出力によって、プログラムを動作させている環境のディスプレイなどから、プログラムが計算した値などを確認することができます。

第1章 INFO1_01 では、標準出力について学びます。

記憶

プログラミングにおいて、メモリ、つまり記憶装置を意識することはとても重要です。メモリは、計算結果を記憶する装置であり、プログラム・プログラマは、どのデータをメモリのどこに記憶しているかを管理する必要があります。メモリの特定領域に名前を付けてプログラムからアクセスできるようにする仕組みを変数と言います。また、連続したメモリ領域を確保して、大規模なデータを扱いやすくする仕組みが配列変数です。

第2章 INFO1_02 では、変数の基本的な概念を学びます。第10章 INFO1_10 と第12章 INFO1_12 では配列の仕組みを学びます。

入力

出力と同様に、コンピュータがデータを読み込む入力元は、キーボード、マウス、ファイル、ネットワークなど様々です。その中でも、最も基本的な入力元が標準入力です。標準入力によって、プログラムを動作させている環境のキーボードから、数値や文字列の形式でデータを入力することができます。

第3章 INFO1_03 では、標準入力の基本的な方法について学びます。また、第11章 INFO1_11 では、1 行から複数のデータを読み込む方法など、標準入力から様々な形状のデータを入力し、それらを目的に合ったデータの形式で扱うための方法を学びます。

計算

コンピュータは計算を行う装置です。計算には、電卓で扱うような最も基本的な四則演算、複数の値を比べるための比較演算、比較演算から得られた真理値を計算対象とする論理演算など、様々な種類があります。特に、比較演算や論理演算は、処理の流れを「制御」する条件を定義するために必要になります。

第4章 INFO1_04 では、四則演算を学びます。第 5 章 INFO1_05 と第6章 INFO1_06 では、それぞれ比較演算と論理演算を学びます。

制御

コンピュータは、電卓のように1回の計算をするだけの単純な道具ではありません。出力データを得るために、記憶と計算を繰り返します。プログラムは、命令が書かれた順番に実行される逐次構造だけでなく、プログラムの実行中のある条件に基づいて処理を分岐する選択構造、条件に応じて同じ処理を繰り返す反復構造を用いて処理を制御します。

第7章 INFO1_07 では、選択構造によって条件に応じて処理を分ける方法を学びます。第8章 INFO1_08 では、反復構造によって条件に応じて処理を繰り返したり、繰り返しの処理を制御する方法を学びます。

プログラミング言語 Python

Python は、近年最も人気のあるプログラミング言語のひとつです。Python は、プログラムの読みやすさ、書きやすさを重視した設計で開発されているため、プログラミングの入門教育にも適した言語となっています。ただし、読みやすい・書きやすいからと言って機能が限られているわけではありません。その応用分野は広く、その内部の処理を理解するためにはより深く学ぶ必要があります(このコースでは、基本文法を学びます)。

Python は、Windows、macOS、Linux などの様々な OS に対応しており、様々なアプリケーションの開発に応用されています。特に、人工知能(AI)、データサイエンス、ロボットなどの分野で広く利用されています。これらの分野に特化したライブラリ(便利なプログラムの集まり)が充実していることも、Python の人気が急激に上昇した要因となっています。

Python の起動・終了方法

Python はインタプリタ言語に分類されます。プログラムを一行ずつ読み込み、逐次解釈しながら実行するソフトウェアまたはツールをインタプリタと言います。一方、プログラムを機械語に翻訳し実行可能なファイルを生成するツールをコンパイラと言います。Python のプログラムは Python インタプリタにより逐次的に読み込まれ、実行されます。

Python のプログラムを実行する方法は以下の 2 通りあります。

1. 対話モードで直接ターミナルにコードを記述して、インタプリタで逐次実行する
2. エディタにソースコードを記述し、ファイルに保存したコードをインタプリタに読み込ませてアプリケーションとして実行する

対話モードでは、実行結果を確認しながら、コードを逐次実行することができます。Windows の場合は、コマンドプロンプトで「python」と入力することで、対話モードを開始することができます。macOS、Linux の場合も、ターミナル上で「python3」と入力することで、対話モードを開始することができます。対話モードはプログラミング入門のツールとして、プログラムの基礎的な概念の確認や簡単な計算機として手軽に活用することができます。

Python のソースコードは、Python インタプリタをインストールしたマシンによってアプリケーションとして実行することができます。また、Google の Colaboratory に代表されるように、オンラインのブラウザ上で Python のプログラムを実行することができます。

プログラムの自動採点を行うオンラインジャッジシステムは、提出されたプログラムをファイルとして記録し、複数のテストデータを用いて実行し採点を行います。そのため、演習問題は、エディタに記述されたソースコードをアプリケーションとして実行する方法で問題を解いていきます。

Python のコメント

本解説では、プログラムの説明のためにコメントを用います。ソースコードには、「コメント」として、実行に影響しない説明文を加えることができます。Python では「#」以降の文字列がコメントとなります(コメントアウトされると言います)。たとえば、次のプログラムにはいくつかのコメントが含まれています。

プログラム

```
# 2つの辺の長さ a と b から長方形の周長と面積を求めます。
a = 15 # 長方形の長辺
b = 9  # 長方形の短辺
print(2*(a+b)  # 長方形の周長
print(a*b)     # 長方形の面積

# ここから
# …
# …
# ここまでは、コメントになります

'''
ここには複数行の
コメントを
書くことができます。
'''
```

#記号の後ろの部分はコメントとして処理され、何を記述しても実行されません。

Python 上では'''や"""で囲まれた部分もコメントとして処理することができますが、実際には複数行テキストとして囲まれている状況になります。

1. 出力

プログラム(ソフトウェア)は、何らかのデータを読み込み、計算をし、その結果を提供します。プログラムの動作を確認する最も基本的な方法が、計算結果を出力しその内容を確認することです。このトピックを修了すれば、何らかのデータを出力できるようになります。

1.1 標準出力

プログラムが計算結果を出力する先を標準出力と呼び、特に指定がない場合は、プログラムを実行しているオペレーティングシステム(OS)のディスプレイ(コマンドプロンプトなど)が該当します。ただし、プログラムの実行方法によっては、標準出力の先がファイルになることもあります。

Python では、print 関数を用いて、数値、文字列、計算結果等を標準出力に書き出すことができます。

関数とは、ある目的を達成するための処理をまとめ、呼び出して使えるようにしたコードのまとまりです。汎用的な関数は、ライブラリ(プログラムの集まり)として準備されており、関数の中身を意識することなく使うことができます。関数には、その処理に関連する名前が付けられており、以下の形式で呼び出します。

関数名(引数)

関数に渡すデータを引数と呼びます。print 関数は、受け取った値を標準出力に出力します。たとえば、次のプログラムは標準出力に1つの整数 86 を出力します。

プログラム	print(86)
出力	86

1.2 リテラル

プログラムは記号や単語からなる文字列の羅列です。プログラムのソースコードに直接記述される数値や文字列をリテラルと呼びます。たとえば、print(86)の 86 は数値リテラルです。「プログラムの要素」とリテラルは、インタプリタやコンパイラが理解できるように区別しなければなりません。ソースコードの中で、プログラムの一部として特定の文字列を使用したい場合は、その文字列を″(ダブルクォーテーション)で囲むか、、(シングルクォーテーション)で囲みます。これは、プログラムの本体に含まれるキーワードと区別するために必要になります。このようにプログラムの中に埋め込まれた文字列を文字列リテラルと呼びます。

たとえば、次のプログラムは標準出力に1つの文字列「hello」を出力します。

プログラム	print("hello")
出力	hello

1.3 演習問題

INFO1_01_A 整数の出力

前解説

print 関数は、引数に指定された値を標準出力に出力します。( )の中に計算式を含めることができます(計算式については INFO1_04 で学習します)。たとえば、以下のプログラムはそれぞれ標準出力に1つの整数を出力します。

プログラム	`print(1)  # 1 を出力` `print(500 + 18)  # 518 を出力`
出力	`1` `518`

問題

1つの整数を標準出力に出力せよ。

入力例	出力例
	3

INFO1_01_B 文字列の出力

前解説

文字列リテラルは対象となる文字列を'(シングルクォーテーション)または"(ダブルクォーテーション)で囲います。これらを混用できないことに注意してください。たとえば、以下のプログラムはそれぞれ標準出力に1つの文字列を出力します。

プログラム	`print("information")`
	`print('technology')`
	`print('T')`

出力	`information`
	`technology`
	`T`

問題

1つの文字列を標準出力に出力せよ。

入力例	出力例
	programming

INFO1_01_C 複数の整数の出力

前解説

print 関数は、出力したいデータの項目をカンマで区切ることで、複数のデータを出力することができます。たとえば、次のプログラムは標準出力に空白区切りで `1 2 3` を出力します。

プログラム	`print(1, 2, 3)`
出力	`1 2 3`

問題

2つの整数を1つの空白区切りで標準出力に出力せよ。

入力例	出力例
	1 3

INFO1_01_D 改行

前解説

print 関数は、特に何も指定しなければ、受け取ったデータを出力した後に1つの改行を出力します。

問題

1つの整数を1行に標準出力に出力せよ。

入力例	出力例
	3

INFO1_01_E 複数の整数の出力

前解説

プログラムの基本構造のひとつが「逐次処理」です。逐次処理では、書かれた命令文が（上から下へ向かって）順番に一つずつ逐次実行されます。基本的には、ある処理を実行するまえに、その直前の処理が終了しています。たとえば、2つの整数 100 と 200 をこの順番にそれぞれ1行に出力するには、以下のように2つの print 関数を順番に実行します。

プログラム	`print(100)` `print(200)`
出力	`100` `200`

問題

3つの整数をそれぞれ1行に標準出力に出力せよ。

入力例	出力例

	`1`
	`2`
	`3`

2. 変数

コンピュータは、入力されたデータや計算結果を一時的にメモリに記憶しながら処理を進めます。このトピックを修了すれば、所定のメモリ領域に値を記憶しそれらを適宜活用できるようになります。

2.1 変数

コンピュータのメモリ上の特定の領域に名前を付けてデータを格納する仕組みを変数と呼びます。変数は、プログラムの中で使われるキーワードで、プログラマが自由に名前を付けることができます。これは、プログラム(プログラマ)が、どのメモリ領域に何のデータを記録して読み書きするかを判断できるように、データの格納場所に名札をつけるようなものです。

変数の命名には以下の通りいくつか決まりがあり、基本的にはアルファベットと数字を用いた文字列になります。この解説では、変数名を簡略化して説明する場合がありますが、ソフトウェアの開発では、そのデータの意味や役割が分かるように変数名を命名することが重要です。

変数の命名規則

- 英小文字、英大文字、数字、アンダースコアのみ使用できる(大文字と小文字は区別される)
- 数字から始めてはいけない
- 予約語(Python の文法で定義されている名前)は使用不可能:`if`, `for`, `and` など

2.2 代入演算

変数の役割は、プログラムで扱う値をメモリへ書き込み、必要な時に参照することです。変数は、代入演算を行う代入演算子 = によって初期化を行い、プログラムの中で使えるようにします。プログラムの中で各種の演算を表す記号を演算子と呼びます。

代入演算では代入演算子 = の左側の変数(メモリ領域)に、右側の計算式の値(計算結果)が書き込まれます。

たとえば、次のプログラムは変数 a に整数 123 を書き込みます。

プログラム	`a = 123`

これは、メモリ上の特定領域(箱のようなイメージ)に a という名前をつけて、その領域に 123 を格納しています。「a と 123 が等しい」という意味ではないことに注意してください。

一方、プログラムの式に記述された変数は、その値が参照されます。たとえば、次のプログラムは変数 a の値を出力します。

プログラム	`a = 123` `print(a)`
出力	`123`

また、たとえば次のプログラムは、変数 a の値を変数 b に代入しその値を出力します。

プログラム	`a = 123`
	`b = a  # b に a の値のコピーを代入する。a の値はそのまま`
	`print(b)  # 123 が出力される`

出力	`123`

2.3 基本データ型

プログラムで扱う値や変数にはそのデータ型があります。データ型とは、整数や文字列といったデータの種類を表す属性です。変数に値が書き込まれたときに、そのデータ型が判別され、その変数のデータ型として管理されます。

Python では、以下の表に示すように、様々なデータ型を扱います。

データ型	列
整数(int)	`-8, 0, 1, 98, 1024, …`
小数(float)	`3.14, -70.86, …`
文字列(str)	`'a', 'abc', '###', "01234", "xyz", 'hello aizu', …`
真偽値(bool)	`True, False`

int は integer を意味し、整数を表すデータ型です。float は浮動小数点数を表すデータ型です。str は string を意味し、文字列を表すデータ型です。bool は boolean を意味し、真理値を表すデータ型です。真理値は、True(真)または False(偽)の値をとります。

2.4 演習問題

INFO1_02_A 変数

問題

以下の手順を実行せよ。

- 変数aに整数を代入する。変数aの値を出力する。
- ただし、代入する整数は 1 から 100 以下とする。

入力例	出力例
	18

INFO1_02_B 変数の上書き

前解説

変数はその名の通り「変わる数」で、代入演算によって値を何度でも変更することができます。たとえば、次のプログラムは変数aの内容を書き換えるプログラムです。

プログラム	`a = 5`
	`print(a)  # 5 が出力される`
	`a = 3  # a の値を 3 に書き変える`
	`print(a)  # 3 が出力される`
	`a = 1  # a の値を 1 に書き変える`
	`print(a)  # 1 が出力される`

出力	5 3 1

問題

以下の手順を実行せよ。

- 変数aに整数を代入する。
- aの値を1行に出力する。
- 変数aに整数を代入する。
- aの値を1行に出力する。

ただし、代入する整数は 0 以上 100 以下とし、2回目に代入する整数は1回目に代入した整数よりも大きいものとする。

入力例	出力例
	18 26

INFO1_02_C 2つの変数

前解説

一般的に、プログラムは複数の異なるデータを保持するために、複数の変数を利用します。たとえば、次のプログラムは、変数 a,b にそれぞれ 5, 8 を格納し、その後それらの中身を出力するプログラムです。

プログラム	a = 5 b = 8 print(a, b)
出力	5 8

問題

以下の手順を実行せよ。

- 変数aに整数を代入する。
- 変数bに整数を代入する。
- aの値とbの値をこの順番に空白区切りで1行に出力する。
- bの値とaの値をこの順番に空白区切りで1行に出力する。

ただし、代入する整数は 0 以上 100 以下とし、aとbに代入する整数は異なるものとする。

入力例	出力例
	1 8 8 1

INFO1_02_D 交換

前解説

変数はそれが指すメモリ領域の値を返します。たとえば、次のプログラムは、変数a, bにそれぞれ 5, 8 を格納し、その後それらの中身を交換し、出力するプログラムです。このような手順をスワップと呼びます。

プログラム	`a = 5`
	`b = 8`
	`t = a  # a の値を別の変数に保持しておく`
	`a = b  # a の値は b の値で上書きされる`
	`b = t  # 保持しておいた値を b に代入する`
	`print(a, b)`

出力	`8 5`

Python では、複数の値を1つの要素としてまとめて扱うことができます。これを Tuple(タプル)と言います。タプルを用いて、変数aと変数bの値を交換する処理は、以下のように書くこともできます。

プログラム	`a = 5`
	`b = 8`
	`a, b = b, a`
	`print(a, b)`

出力	`8 5`

このプログラムの 3 行目では、変数a, bに変数b, aの値がそれぞれこの順番で代入されます。

問題

以下の手順を実行せよ。

- 変数aに整数を代入する。
- 変数bに整数を代入する。
- aの値とbの値をこの順番に空白区切りで1行に出力する。
- aとbの値を交換する。
- aの値とbの値をこの順番に空白区切りで1行に出力する。
  ただし、代入する整数は 0 以上 100 以下とし、aとbに最初に代入する整数は異なるものとする。

入力例	出力例
	`1 8` `8 1`

3 入力

コンピュータは、何らかのデータを読み込み、その入力に基づいた計算結果を提供する装置です。入力されたデータはメモリ(変数)に記録されることで、計算の準備が整います。このトピックを修了すれば、入力データに応じた出力が行えるようになります。

3.1 標準入力

プログラムが受け取るデータの入力元を標準入力と呼び、特に指定がない場合は、コンピュータのキーボードになります。ただし、プログラムの実行方法によっては、ファイルの内容が標準入力になることもあります。

データの入力先は主にプログラム中の変数(実行時のメモリ)になります。

Python では、input 関数で標準入力を行うことができます。input 関数は、標準入力から1行の文字列を入力します。

たとえば、標準入力に入力された文字列をそのまま標準出力に出力するプログラムは以下のようになります。

プログラム	
`a = input()` `print(a)`	
入力	出力
2023	2023

このプログラムは、変数を用いずに以下のように書くこともできます。

プログラム	
`print(input())` # `input()` で入力された文字列をそのまま出力する	
入力	出力
2023	2023

3.2 演習問題

INFO1_03_A やまびこ 1

問題

標準入力から1つの整数aが1行に与えられる。aの値を出力せよ。

入力例	出力例
32	32

INFO1_03_B やまびこ 2

前解説

input 関数は、標準入力から1行の文字列を読み込みます。この文字列には空白文字も含まれ、input 関数は改行が現れるまでの1行を読み込みます。

たとえば、次のプログラムは3行に渡る文字列を入力し、それらをそのまま出力します。

プログラム

```
a = input()
b = input()
c = input()
print(a)
print(b)
print(c)
```

入力	出力
red blue yellow	red blue yellow

処理の流れの基本は逐次処理です。上のプログラムは、最初に1行に与えられたデータを変数 a に代入し、その後に1行に与えられたデータを変数 b に代入し、その後に1行に与えられたデータを変数 c に代入し、その後これらのデータをそれぞれ順番に出力しています。

上のプログラムは、変数を使わずに次のように書くこともできます。

プログラム
`print(input())` `print(input())` `print(input())`

問題

1 行目に整数a、2 行目に整数bがそれぞれ与えられる。aの値とbの値を順番に出力せよ。

入力例	出力例
1 2	1 2

INFO1_03_C やまびこ 3

問題

与えられた3つの整数を逆順に出力せよ。

入力例	出力例
1 2 3	3 2 1

INFO1_03_D やまびこ 4

前解説

input 関数は、空白を含む 1 行を 1 つの文字列として読み込むため、1 行に複数のデータが与えられる場合には工夫が必要になります。Python の文字列には、split メソッドと呼ばれる機能によって、文字列を空白で区切った要素のリストへ変換することができます。リストとは、データの列を管理するデータ構造です(リストについては、INFO1_09 で詳しく学びます)。たとえば、次のプログラムは 1 行に与えられた3つの文字列をそれぞれ変数 a, b, c に代入します。

プログラム	

`a, b, c = input().split()`
`print(a)`
`print(b)`
`print(c)`	

入力	**出力**

`1 10 100`	`1`
	`10`
	`100`

`take it easy`	`take`
	`it`
	`easy`

ここで、a, b, c のデータ型はいずれも文字列になります。

問題

空白区切で4つの整数a, b, c, dがこの順番に１行に与えられる。4つの整数 d, c, b, aの値をこの順番に出力せよ。

入力例	出力例
`1 2 3 4`	`4 3 2 1`

INFO1_03_E やまびこ 5

問題

１行目に3つの整数a, b, c、2行目に2つの整数d, eが与えられる。e, d, c, b, a をこの順番で出力せよ。

入力例	出力例
`1 1 1` `2 2`	`2 2 1` `1 1`

4 四則演算

プログラミングの目的はコンピュータに計算をさせることです。計算の最も基本となるものが加算、減算、乗算などの算術演算です。また、計算を行うためには、式の中で使われる変数や値の「型」を意識し、必要に応じて値の型を変換しなければなりません。このトピックを修了すれば、基本的な演算記号を用いて、計算式が記述できるようになります。

4.1 算術演算

プログラムでは、算術演算は算術演算子によって行います。算術演算子は、各種の演算を指示する記号です。Python では以下の表に示す算術演算子を使うことができます。

演算子	意味	プログラムの例	結果の例
+	加算	5 + 3	8
-	減算	5 - 3	2
*	乗算	5 * 3	15
/	除算	5 / 3	1.66666667
//	切り捨て除算	5 // 3	1
%	剰余(余り)	5 % 3	2
**	累乗	5 ** 3	125

これらの演算は２項演算と呼ばれ、演算子の左の式の値と右の式の値を参照して計算を行います。最も単純な式は変数ひとつ、あるいは値ひとつとなるので、たとえば、「x + 2」は、変数 x の値と値２を加算してその結果を返します。一般的に、値、変数、演算子等を組み合わせて値を生成するものを式と呼びます。たとえば、「a」、「a + 1」、「2 * x + 10」などはいずれも式になります。

代入演算子 = は、左辺の変数に右辺の式の値(計算結果)を代入します。たとえば、次のプログラムは、変数 x の値と８の和を変数 y に代入します。

プログラム

```
y = x + 8
```

「y = x + 8」が「y は (x+8) に等しい」という意味にはならないことに注意してください。

4.2 演算子の優先度

複数の演算子が含まれる式が評価されるときは、演算子の優先度に基づいて計算が実行されます。ここで、演算や式を実行して値を得ることを「評価」と呼びます。複数の 算術演算が含まれる式では、乗算・除算が加算・減算よりも優先され評価されます。

たとえば、式 `x + y * z` では、`y * z` が先に計算されその結果と `x` の和が計算されます。

( )を用いて、式の中の演算の優先度を明示することができます。

たとえば、式 `(x + y) * z` では、`x + y` が先に計算されその結果と `z` の積が計算されます。

4.3 キャスト

プログラムの中で、文字列を数値に変えるなど、値のデータ型を変換したい場合があります。データ型を変換する操作をキャストと呼びます。Python では、主に以下の関数を用いることでキャストを行うことができます。

関数	意味	プログラムの例	結果の例
`int`	整数へキャスト	`int(3.14)  # 浮動小数点数から整数` `int('1234')  # 文字列から整数`	`3` `1234`
`str`	文字列へキャスト	`str(777)  # 整数から文字列`	`"777"`
`float`	浮動小数点数へキャスト	`float(18)  # 整数から浮動小数点数` `float("3.14")  # 文字列から浮動小数点数`	`18.0` `3.14`

input 関数が標準入力から文字列を受け取った際、そのデータ型は文字列型となります。標準入力からの文字列を、数値として扱う場合は、それを数値にキャストする必要があります。

たとえば、標準入力から1つの整数を読み込み変数 `x` に代入するには、以下のように記述します。

プログラム	
`x = int(input())  # x を整数として初期化する` `print(x + 1)  # x に 1 を加算した値が出力される`	
入力	**出力**
`100`	`101`

このようなキャスト演算により、変数 x は整数として扱えるようになります。一方、たとえば以下のようにキャストを行わずに数値を加算しようとするとエラーが発生します。

プログラム	
`x = input()  # x は文字列型` `print(x + 1) # 文字列に数値の 1 を加算しようとしているのでエラー`	
入力	出力
100	エラー!!

4.4 演習問題

INFO1_04_A 和

問題

2つの整数A, Bの和$A + B$を計算せよ。

入力例	出力例
1 # A 2 # B	3

INFO1_04_B 差

問題

2つの整数A, Bの差$A - B$を計算せよ。

入力例	出力例
1 2	-1

INFO1_04_C 積

問題

2つの整数A, Bの積$A \times B$を計算せよ。

入力例	出力例
2 3	6

INFO1_04_D 商

前解説

//演算子または / 演算子で除算を行います。
a//b は、a を b で割った商を整数で返します。一方、a/ bは、a を b で割った値を浮動小数点数で返します。
たとえば、以下のプログラムは割り算の結果を出力するプログラムです。

プログラム

`a = 7`
`b = 3`
`print(a // b)`
`print(a / b)`

出力
`2` `2.3333333333333335`

問題

2つの整数A, Bの商$A \div B$を計算せよ。ただし、小数点以下を切り捨てること。

入力例	出力例
7 3	2

INFO1_04_E 剰余

問題

2つの整数A, Bについて、AをBで割った余りを計算せよ。

入力例	出力例
7 3	1

INFO1_04_F 累乗

前解説

累乗は、掛け算の演算子または累乗の演算子で記述することができます。掛け算で行う場合、たとえば、`x` の3乗は `x * x * x` と記述することができます。
累乗の演算子は `**` で表され、左側が底、右側が指数となります。たとえば、`x**3` は `x` の３乗を返します。たとえば、次のプログラムは `x` の４乗を計算します。

プログラム

`x = 2`
`print(x * x * x * x)`
`print(x**4)`

出力
`16` `16`

問題

整数Aの10乗を計算せよ。

入力例	出力例
`2`	`1024`

INFO1_04_G 計算式1

問題

3つの整数A, B, Cが与えられる。$A - B \times C$の値を求めよ。

入力例	出力例

`1 # A`
`2 # B`
`3 # C`
	`-5`

INFO1_04_H 計算式2

問題

5つの整数$\mathrm{A, B, C, D, E}$が与えられる。$\mathrm{A} \times \mathrm{B}^3 + \frac{\mathrm{C} \times \mathrm{D}}{\mathrm{E}} - 100$の値を求めよ。

入力例	出力例
3 # A 2 # B 8 # C 3 # D 2 # E	-64

INFO1_04_I 計算式3

問題

2つの整数A, Bが与えられる。$A - (\frac{A}{B} \times B)$求めよ。ただし、$\frac{A}{B}$は、$A$を$B$で割った商とする。

入力例	出力例
2 1	0

5 比較・等値

既に学習したように、コンピュータが行う最も基本的な演算が算術演算です。変数や値、算術演算を組み合わせた式によって目的の値を計算します。一方、プログラムは単なる式の値を求めるだけではなく、条件を満たすかどうかを判断して処理を分岐させることによって様々な計算を行います。この判断のための基本的な演算が、2つの式の値を比べることです。このトピックを修了すれば、2つの式の値の等価性や大小関係を比べることができるようになります。

5.1 真理値

「真」(True)であるか「偽」(False)であるかを示す値を真理値と言います。真理値を扱うデータ型が Boolean 型です。Python では、真理値のリテラルは `True` と `False` になります。

5.2 等値・非等値演算子

等値・非等値演算子は、2つの式の値が等しいか否かを判定する演算子です。等値・非等値演算子を以下の表に示します。

演算子	数学記号	例	結果
`==`	=	`A == B`	式 A の値と式 B の値が等しいとき True となる
`!=`	≠	`A != B`	式 A の値と式 B の値が等しくないとき True となる

等値・非等値演算子を用いた具体例を以下の表に示します。

演算子	プログラムの例	結果

`==`	`5 == 5`
	`x = 3`
	`y = 4`
	`x == y`
		`True`
		
		`False`

`!=`	`5 != 5`
	`x = 3`
	`y = 4`
	`x != y`
		`False`
		
		`True`

5.3 比較演算子

比較演算子は、2つの式の値の大小関係を判定する演算子です。比較演算子を以下の表に示します。

演算子	数学記号	例	結果
<	＜	A < B	式Ａの結果が式Ｂの結果より小さいとき True
>	＞	A > B	式Ａの結果が式Ｂの結果より大きいとき True
<=	≦	A <= B	式Ａの結果が式Ｂの結果以下のとき True
>=	≧	A >= B	式Ａの結果が式Ｂの結果以上のとき True

プログラムでは≦, ≧ が使えないため、それぞれ <=, >= と記述します。右側に = が書かれることに注意してください。

比較演算子を用いた具体例を以下の表に示します。

演算子	プログラムの例	結果
<	5 < 10	True
>	5 > 10	False
<=	5 <= 4	False
>=	5 >= 5	True

5.4 演算子の優先度

等値演算・不等値演算・比較演算よりも、算術演算の方が優先的に計算されます。

たとえば、具体例を以下の表に示します。

演算子	プログラムの例	結果
==	7 == 3 + 4	７と３＋４の計算結果が比較され True となる
!=	2 * 3 + 1 != 7	２＊３＋１の結果と７が比較され False となる
<	1 + 2 + 3 < 4 + 5	６＜９と評価され True となる
>	a = 2 b = 3 2 * a + 1 > b + 1	５＞４と評価されて True となる

5.5 演習問題

INFO1_05_A 等値

前解説

等値演算子 == は2つの値が等しい場合に真(True)、等しくない場合に偽(False)を返します。代入演算子 = と異なり、= を2つ続けることに注意して下さい。

たとえば、次のプログラムは等値演算を行い、結果を出力します。

プログラム

```
a = 1
b = 2
c = 1
print(a == b) # False
print(a == c) # True
```

出力

```
False
True
```

問題

2つの整数A, Bが与えられる。AとBが同じ値かどうかを判定せよ。

入力例	出力例
1 1	1
1 2	0

INFO1_05_B 非等値

前解説

不等値演算 != は2つの値が異なる場合に真(True)、等しい場合に偽(False)を返します。たとえば、次のプログラムは不等値演算を行い、結果を出力します。

プログラム

`a = 1`
`b = 2`
`c = 1`
`print(a != b) # True`
`print(a != c) # False`

出力
`True` `False`

問題

2つの整数A, Bが与えられる。AとBが異なる値かどうかを判定せよ。

入力例	出力例
`1 # A` `1 # B`	`0`
`1` `2`	`1`

INFO1_05_C 比較 1

前解説

比較演算子は2つの値の大小関係を評価します。たとえば、次のプログラムは変数の値を比較します。

プログラム

```
S = 1
M = 1
L = 2
print(S <= M)  # True
print(S < M)   # False
print(L > S)   # True
print(L >= S)  # True
```

出力

```
True
False
True
True
```

問題

2つの整数A, Bが与えられる。AがBより小さいかどうかを判定せよ。

入力例	出力例
1 2	1
2 2	0

INFO1_05_D 比較 2

問題

2つの整数A, Bが与えられる。AがB以下かどうかを判定せよ。

入力例	出力例
1 2	1
2 2	1

INFO1_05_E 評価式1

問題

2つの整数A, Bが与えられる。AとBの和が 50 以上かどうかを判定せよ。

入力例	出力例
30 40	1
20 25	0

6 論理演算

等値演算子や比較演算子を用いれば、2つの式の値に対する比較を行ってその状態を判定することができます。一方、より複雑な判断を行うためには、いくつかの状態を組み合わせて結果を判定する必要があります。このトピックを修了すれば、複数の条件を組み合わせた式によって、状態を判定できるようになります。

6.1 論理演算子

真(True)と偽(False)の 2 種類の値、つまり真理値に対して行う演算を論理演算と呼びます。論理演算の結果も真理値になります。以下の表に示すように、代表的な論理演算には、「論理積」(AND)、「論理和」(OR)、「論理否定」(NOT)などがあります。

演算子	演算	数学記号	意味	例
not	論理否定	¬	否定	not P
and	論理積	∧	かつ	P and Q
or	論理和	∨	または	P or Q

与えられた命題 P, Qについて、論理演算を適用するとその結果は以下の表のようになります。

P	$\neg P$
真	偽
偽	真

P	Q	$P \wedge Q$
真	真	真
真	偽	偽
偽	真	偽
偽	偽	偽

P	Q	$P \vee Q$
真	真	真
真	偽	真
偽	真	真
偽	偽	偽

論理演算子を用いた具体例を以下の表に示します。

演算子	具体例	結果
not	not(5 == 5)	5 == 5 が True となり、その否定で False
and	(5 == 5) and (10 == 10) (5 <= 5) and (10 <= 5)	5 == 5 が True、10 == 10 が True となり、True 5 <= 5 が True、10 <= 5 が False となり、False
or	(5 == 5) or (10 == 10) (5 <= 5) or (10 <= 5)	5 == 5 が True、10 == 10 が True となり、True 5 <= 5 が True、10 <= 5 が False となり、True

6.2 論理演算の優先度

論理演算に用いられる真理値は、変数や式から得られます。算術演算、等値・否等値・比較演算、論理演算の中では、算術演算が最も優先度が高く、等値・否等値・比較演算、論理演算の順に低くなります。つまり、論理演算の優先度が最も低くなります。たとえば、以下の3つの式は同等で同じ真理値を返します。

プログラム

```
a + b == c and x > y * z
(a + b == c) and (x > y * z)
((a + b) == c) and (x > (y * z))
```

これらの式は「a＋b の値が c に等しく、かつ x の値が y と z の積よりも大きい」とき True を返します。

6.3 論理演算の結合

P、Q、R をそれぞれ真理値とすると、「P または Q または R」は以下のように記述でき、論理演算を連結することができます。

```
P or Q or R
```

これは以下のプログラムと同等で、ある論理演算の結果が別の論理演算に用いられるととらえることもできます。

```
(P or Q) or R
```

論理演算子の間の優先順位は、否定が最も高く、論理積、論理和の順に低くなります。

たとえば、「P または Q かつ R」は以下のように記述することができ、論理積 and が先に計算されます。

```
P or Q and R
```

これは以下のプログラムと同等です。

```
P or (Q and R)
```

また、たとえば、「P かつ Q ではない」は以下のように記述することができ、論理否定 not が先に計算されます。

```
P and not Q
```

これは以下のプログラムと同等です。

```
P and (not Q)
```

6.4 演習問題

INFO1_06_A NOT

前解説

真理値Pの否定、つまり「Pではない」は次のように記述します。

```
not P
```

演算子 not はPの否定を真理値として返します。

たとえば、次のプログラムは真理値の否定を出力します。

プログラム

```
P = True
Q = False
print(not P) # False
print(not Q) # True
```

出力

```
False
True
```

問題

真理値Aの否定を求めよ。

入力例	出力例
0	1
1	0

INFO1_06_B AND

前解説

「PかつQ」を表す演算を論理積と呼び、and 演算子で計算します。たとえば、

```
P and Q
```

は真理値Pと真理値Qがともに True のとき True を返します。つまり、PとQのうちいずれかが False のとき False を返します。

たとえば、次のプログラムは論理積の結果を出力します。

プログラム

```
P = True
Q = False
print(P and P) # True
print(P and Q) # False
print(Q and P) # False
print(Q and Q) # False
```

出力

```
True
False
False
False
```

問題

4つの整数A, B, C, Dが与えられる。「AとBが等しい」かつ「CとDが等しい」が成り立つかどうかを判定せよ。

入力例	出力例
10 # A 10 # B 20 # C 20 # D	1
10 10 20 30	0

INFO1_06_C OR

前解説

「PまたはQ」を表す演算を論理和と呼び、or 演算子で計算します。たとえば、

```
P or Q
```

は真理値Pと真理値Qのいずれかが True のとき True を返します。つまり、PとQがともに False のとき False を返し、それ以外の場合は True を返します。

たとえば、次のプログラムは論理和の結果を出力します。

プログラム

```
P = True
Q = False
print(P or P) # True
print(P or Q) # True
print(Q or P) # True
print(Q or Q) # False
```

出力

```
True
True
True
False
```

問題

4つの整数A, B, C, Dが与えられる。「AとBが等しい」または「CとDが等しい」が成り立つかどうかを判定せよ。

入力例	出力例
10 10 20 30	1
10 20 30 40	0

INFO1_06_D 論理式1

問題

整数Aが与えられる。Aが 2, 3, 5,または 7 であるかどうかを判定せよ。

入力例	出力例
3	1
6	0

INFO1_06_E 論理式2

問題

4つの整数A, B, C, Dが与えられる。「AとBが等しくかつCとDが等しい」または「AとCが等しくかつBとDが等しい」かどうかを判定せよ。

入力例	出力例
1 1 2 2	1
1 2 1 2	1
1 2 3 4	0

INFO1_06_F 論理式3

問題

A, B, Cの真理値が与えられる。「”AかつB”が成り立たない」または「Cが成り立つ」かどうかを判定せよ。

入力例	出力例
1 # A 0 # B 0 # C	1
1 1 1	1
1 1 0	0

7 条件分岐

変数と四則演算を用いれば、異なる入力に対して目的の計算結果を得ることができます。一方、普通の計算機と違ったプログラミングの強みは、条件によって計算式やまとまった処理の内容を変えたり、処理を繰り返したり、あるいは処理を停止できることです。このトピックを修了すれば、条件によって処理を変えることができるようになります。

7.1 構造文

計算式の結果を変数に代入する処理は、1つの「文」として記述され実行されます。文のまとまりは複合文、あるいはブロックと呼ばれます。プログラミング言語によっては { } で囲まれた部分をブロックとしますが、Python ではインデント(字下げ)を使用してブロックを表現し、同じ幅の字下げされたコードが同じブロックに属するようにプログラムを記述します。

処理の流れを制御する文は構造文と呼ばれ、条件を指定した後にブロックが記述されます(より厳密な定義は、トピック INFO1_08 で確認します)。

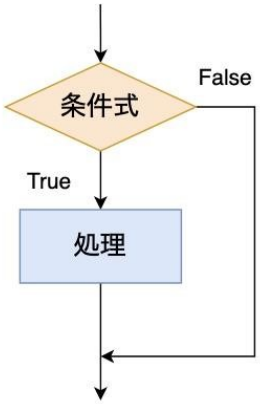
7.2 条件分岐

プログラムは、書いた文がそれらの順番に実行される逐次処理によって処理が進みます。一方、条件によってプログラムを分岐させ、実行する処理を変えることによって様々な処理を実現します。たとえば、以下のような分岐処理が考えられます。

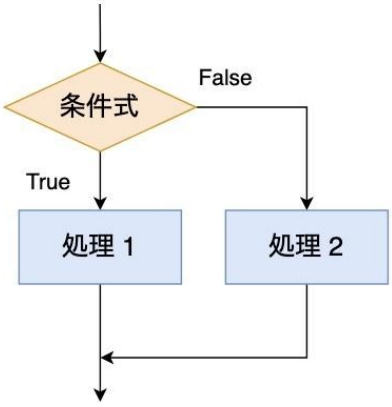
- 得点 P が 50 点以上であれば合格、50 点未満であれば不合格とする
- 得点 P が 80 点以上 100 点以下であれば、成績 G を'A'とする
- 得点 P が 50 点未満または出席回数が 10 回未満であれば、不合格とする

処理の流れを制御する最も基本的な構造文は、条件に応じて異なる処理を実行させることができる条件分岐です。条件分岐には、主に if 文、if / else 文、if / elif / else 文があります。

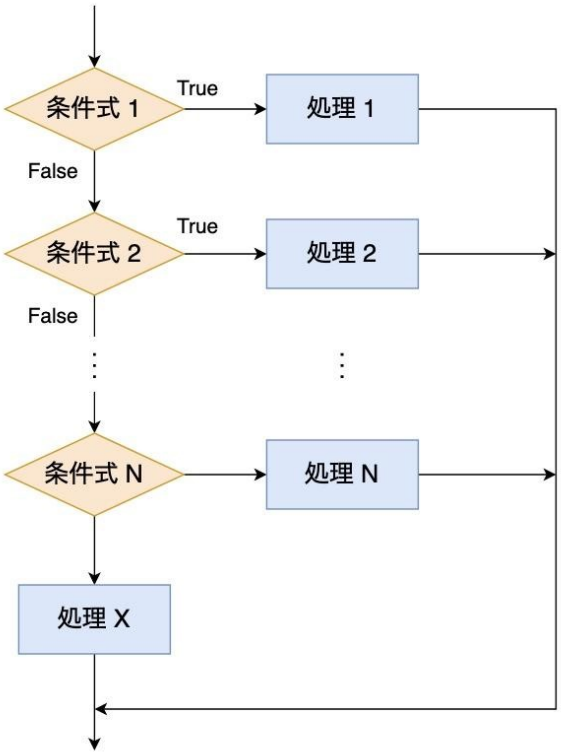
if 文は、下図のように「条件式の値が True であれば処理を実行し、False の場合は何もしない」という流れを実装します。



if / else 文は、下図のように「条件式の値が True であれば処理1を実行し、False であれば処理2を実行する」という流れを実装します。



if / elif / else 文は、下図のように「条件式1が True であれば処理1を、条件式1が False で条件式2が True であれば処理2を、・・・条件式 N-1 が False で条件式 N が True であれば処理 N を、そうでなければ処理 X を実行する」という流れを実装します。


条件式 1
True
処理 1
False
条件式 2
True
処理 2
False
条件式 N
処理 N
処理 X

7.3 演習問題

INFO1_07_A 分岐1

前解説

指定した条件が成り立つとき(真)と、成り立たないとき(偽)で異なる処理を実行したいときには、条件分岐を使います。if 文は最も基本的な条件分岐の構造で以下のように記述します。

```
if 条件式:
    処理
```

このプログラムでは、条件式による真理値が真のときだけ、「処理」が実行されます。処理は改行をしなくとも、:の後に続けて書くことができますが、そのときは処理の内容を1行に書かなければなりません。

処理は、if 条件式:の行よりも一つ多くインデントし記述します。これは Python のプログラムがインデント(字下げ)を含めて解釈されるため必要になります。処理には、いくつでも文(行)を書くことができますが、同じ幅のインデントを加えて書きます。たとえば、以下のプログラムは、インデントによって実行する処理が制御されていることを表します。

プログラム

```
a = 10

if a == 10:
    print("10")
    print("---")

if a == 11:
    print("11")
print("---")
```

出力

```
10
---
---
```

この例は、最後の print("---")が if 文のブロックの外にあるため、条件にかかわらず必ず実行されることを表しています。

問題

与えられた整数xが自然数かどうかを判定せよ。ただし、0 は自然数に含めない。

入力例	出力例
7	yes
-10	

INFO1_07_B 分岐2

前解説

if 文にはいくつかの構造があります。「条件が成り立つなら処理1を実行し、成り立たないなら処理2を実行する」という処理をする if/else 文は以下のように記述します。

```
if 条件式:
    処理1
else:
    処理2
```

このプログラムでは、条件式による真理値が真のとき「処理1」が実行され、偽のとき「処理2」が実行されます。

処理1については、if 条件式:の行よりも一つ多くインデントします。処理2については、else:の行よりも一つ多くインデントします。処理1と処理2にはそれぞれ、いくつでも文(行)を書くことができますが、同じ幅のインデントを加えて書きます。

また、処理1と処理2は改行をしなくとも書くことができますが、そのときは処理をそれぞれ1行に書かなければなりません。

問題

与えられた整数xが自然数かどうかを判定せよ。ただし、0 は自然数に含めない。

入力例	出力例
7	Yes
-10	No

INFO1_07_C 分岐3

前解説

3通り以上の条件分岐を行いたい場合には、多重条件分岐を使います。多重条件分岐は if / elif / else 文を用いて以下のように記述します。

```
if 条件式1:
    処理1
elif 条件式2:
    処理2
...
elif 条件式N:
    処理N
else:
    処理X
```

この多重条件分岐では、条件式1から条件式Nまでを順番に調べていき、最初に成り立った条件式に対応する処理を実行して条件分岐処理を終了します。たとえば、条件式1が成り立てば処理1を実行し、残りの条件式はチェックせずに条件分岐処理を終了します。また、たとえば、条件式1も条件式2も成り立たず、条件式3が成り立つなら、処理3を実行して条件分岐処理を終了します。

また、どの条件式も成り立たなかった場合の処理が不要であれば、`else:`を含んだそれ以降の記述は省略できます。

問題

与えられた整数xの正負を判定せよ。

入力例	出力例
7	1
-10	-1
0	0

INFO1_07_D 分岐4

問題

得点xに対応する成績を求めよ。xが 80 以上 100 以下、65 以上 80 未満、50 以上 65 未満、35 以上 50 未満、0 以上 35 未満のとき、それぞれ A, B, C, D, F と出力せよ。

入力例	出力例
86	A
65	B
30	F

INFO1_07_E 選択1

問題

与えられた2つの整数の最小値を求めよ。

入力例	出力例
2 3	2
10 -7	-7

INFO1_07_F 選択2

問題

与えられた3つの整数の最小値を求めよ。

入力例	出力例
1 2 3	1

8 ループ

演算子を用いた式や分岐構造を用いることで、条件によって様々な計算を処理することができます。これらの逐次処理によって、複数の計算を実行させることもできます。一方、コンピュータが得意としている処理は計算を繰り返す反復処理です。反復処理が行えるようになれば、さらにコンピュータの計算能力の恩恵を受けることができるようになります。このトピックを修了すれば、条件に応じて所定の処理を繰り返すことができるようになります。

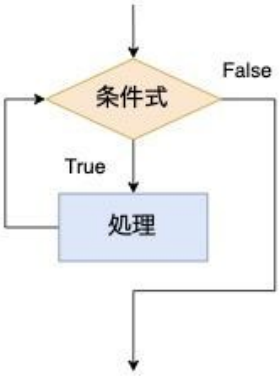
8.1 繰り返し処理

プログラムは、繰り返し構造(これをループといいます)を使い、指定した条件を満たす限り、指定した処理を何度も繰り返すことができます。繰り返し構造は、コンピュータの処理能力を活かして膨大な量の計算を行う強力な構文です。たとえば、以下のような処理が可能になります。

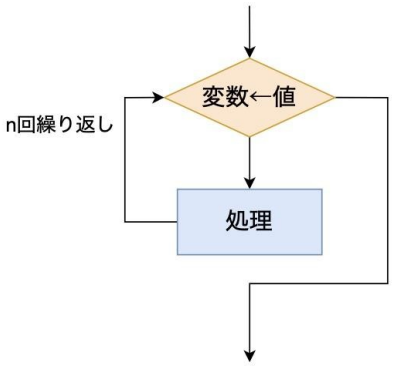
- $a_{1,}a_{2,,...}a_{10000}$の合計値を計算する
- サイコロをふって(1 から 6 までのランダムな数を生成して 1 が出るまで、サイコロをふり続ける)

Python では主に while ループと for ループの2つの種類の構造文を使います。

while ループは下図のように、「条件式が True の間、処理を実行することを繰り返す」という流れを実装します。



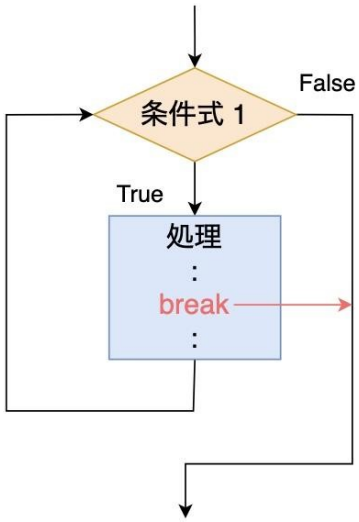
for ループは、「n 回、処理を実行することを繰り返す」という流れを実装します。Python では、リストの各要素を対象として処理を繰り返すことができ、下図のように「すべての要素を使い切るまで、リストの要素の値を順番に変数に入れ、その値を用いながら処理を繰り返す」という流れを実装します。



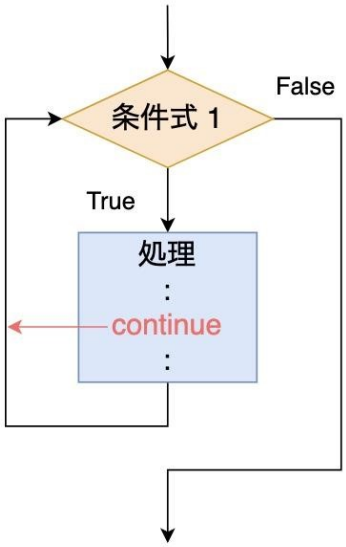
8.2 繰り返し処理の制御

繰り返し処理の流れは、基本的には条件に従いますが、その条件に限らず強制的に流れを制御することもできます。流れの制御には、ループ構造を強制的に終了する命令と、処理を中断して次の回にスキップする命令があります。

break 文は下図のように、繰り返し処理の条件式に関わらず、処理の途中でループ構造から強制的に抜け出したい場合に実行します。while 文または for 文のどちらでも使用することができます。break 文は、break 文が含まれる最も内側の処理をその時点で終了させて、繰り返し処理から抜け出す命令文です。



continue 文は、下図のように continue 文の位置から、それが含まれる最も内側の繰り返し処理の条件式の評価に戻るための命令文です。つまり、現在の処理を中断して、次の回へスキップします。



8.3 複合代入演算子

繰り返し処理には、1回のループが終了するごとに更新される変数を用いる場合があります。このような変数は、ループの制御に用いられる場合もあります。

ある変数に対する演算の結果を同じ変数に代入したい場合は、複合代入演算子を使うことでより簡潔に記述することができます。複合代入演算子は、ループ処理に限らず、主に変数の値を更新する場合に使われます。

複合代入演算子とそれらの利用例を以下の表に示します。

演算子	使用例	意味
`+=`	`i += 1`	`i = i + 1`
`-=`	`i -= 1`	`i = i - 1`
`*=`	`i *= 2`	`i = i * 2`
`/=`	`i /= 2`	`i = i / 2`
`%=`	`i %= 2`	`i = i % 2`
`**=`	`i **= 2`	`i = i ** 2`
`//=`	`i //= 2`	`i = i // 2`

たとえば、`i += 1` は `i = i + 1` と同じ処理であり、変数 i の値に 1 を加算した値が、変数 i に代入されます。

8.4 演習問題

INFO1_08_A N 回繰り返し 1

前解説

while 文は、指定された条件式が満たされる限り、処理を繰り返す構造文です。while 文は以下のように記述します。

```
while 条件式:
    処理
```

条件分岐の構造と同様に、while 条件式:の直後から、等幅でインデントされている行が、対応する繰り返し処理を行う範囲になります。 たとえば、次のプログラムは print 関数の実行を 10 回繰り返します。

プログラム

```
i = 0
while i < 10:
    print('ok')
    i = i + 1
```

出力

```
ok
ok
:
ok
```

このプログラムでは、変数 i の値が初期値の 0 から、処理を1回実行するたびに 1, 2, ..., 10 のように増加します。i の値が 10 に達し i < 10 の条件式を満たさなくなったときに処理が終了します。つまり、処理は i の値が 0, 1, 2, ..., 9 のときに実行され、合計 10 回実行されます。

問題

単語WをN回出力せよ。

入力例	出力例
4 # N	hello hello hello hello

INFO1_08_B N 回繰り返し 2

前解説

繰り返す回数が決まっている場合や特定の規則に従って繰り返す場合は、while 文よりも for 文を用いた方が読みやすくプログラムを書くことができます。

range 関数は、与えられた引数に基づいて数列を生成します。また in 演算子は数列の要素を順番に読み込みます。for 文では以下のように、range 関数で生成した数列の要素を in 演算子によって指定の変数に代入することで、その変数の値に対する処理を順番に(繰り返し)実行します。

```
for 変数名 in range(変数の最初の値，変数の最後の値，変数の増分)：
    繰り返したい処理
```

for 文では、「変数の最初の値」から各処理の後に「変数の増分」を加算していき、「変数の最後の値」未満まで処理を繰り返します。

for を含む文の直後から等幅でインデントされている行が、対応する繰り返し処理を行う範囲となります。

range 関数では、変数の増分や変数の最初の値を省略して書くことができます。次のように、変数の増分を省略した場合は、増分に 1 を指定したときと同じ動作になります。

```
for 変数名 in range(変数の最初の値，変数の最後の値)：
    繰り返したい処理
```

次のように、変数の最初の値と変数の増分を省略した場合は、最初の値は 0 で増分に 1 を指定したときと同じ動作になります。

```
for 変数名 in range(変数の最後の値)：
    繰り返したい処理
```

たとえば、以下の2つのプログラムは同じ動作をし、それぞれ 0 から N-1 までの値を出力します。

プログラム
`for i in range(0, N):` `  print(i)`
`for i in range(N):` `  print(i)`

問題

1からNまでの整数をそれぞれ順番に出力せよ。

入力例	出力例
`5 # N`	1 2 3 4 5

INFO1_08_C 無限ループ 1

前解説

無限ループとは、while 文または for 文の条件式を常に真になるよう指定することで、処理を回し続けるプログラミングテクニックです。

繰り返し処理の制御:break 文

break 文は繰り返し処理の条件式に関わらず、処理の途中でループから強制的に抜け出したい場合に実行します。while 文または for 文のどちらでも使用することができます。break 文は、break 文が含まれる最も内側の処理をその時点で終了させて、繰り返し処理から抜け出す命令文です。

入力値 x が 0 になったところで繰り返し処理を強制終了するには、たとえば、以下のように無限ループと break 文を組み合わせます。

プログラム

```
while True:
    x = int(input())
    if x == 0:
        break
    # 残りの処理
```

問題

与えられた整数を出力する「やまびこ」を反復せよ。入力が 0 のとき入力の終了とする。

入力例	出力例
5 -10 2 -7 6 0	5 -10 2 -7 6

INFO1_08_D 無限ループ 2

前解説

繰り返し処理の制御:continue 文

continue 文は、continue 文の位置から、それが含まれる最も内側の繰り返し処理の条件式の評価に戻るための命令文です。

たとえば、次のプログラムは continue 文を応用して、1 から 10 までの奇数を出力します。

プログラム

```
for i in range(1, 10):
    if i % 2 == 0:
        continue
    print(i)
```

出力

```
1
3
5
7
9
```

continue 文は、繰り返し処理の途中で例外的な状況を判断して残りの処理をスキップしたい場合などに用います。プログラムを読みやすくする上でも応用できるテクニックです。

問題

与えられた整数を出力する「やまびこ」を反復せよ。ただし、負の数は出力しない。

入力例	出力例
5 -10 2 -7 6 0	5 2 6

INFO1_08_E 多重ループ 1

前解説

繰り返し処理は入れ子構造にすることができます。あるループ処理の中に、もうひとつのループ処理を入れた構造を、2重ループと呼びます。より一般的に、いくつかの繰り返し処理を入れ子にした構造を多重ループと呼びます。

たとえば、次のプログラムは2重ループで変数 i と j の値を出力します。

プログラム

```
for i in range(3):
    for j in range(3):
        print(i, j)
```

出力

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

問題

$N \times N$個の#で、正方形を描け。

入力例	出力例
3 # N	### ### ###
4	#### #### #### ####

INFO1_08_F 多重ループ 2

問題

各値a_iをa_i個の#で表したヒストグラムを描け。

入力例	出力例
3 # データの数 5 8 3	##### 5 ######## 8 ### 3

INFO1_08_G 構造化1

前解説

プログラムは、「実行文」と呼ばれる文(statement と呼ばれます)の列によって処理が記述されます。実行文は、以下の3つの文に分類されます。

- 単純文:改行や; (セミコロン) で終了する1つの命令文
- 複合文:中に複数の「文」が含まれる複合的な文。複合文はブロックとも呼ばれる
- 構造文:if 文、for 文、while 文などの制御構造で、条件式などが書かれた直後の「文」には1つの「実行文」が含まれる

構造文が実行文で構成されるように、これらの文は、入れ子構造にすることができます。つまり、実行文の中に別の「文」を含めてプログラムを構成することができます。例えば、if 文の中にさらに if 文や for 文などを含めたり、for 文の中にさらに if 文や while 文などを含めることができ、2重、3重・・・のループ構造も実現することができます。

一般的にプログラムは以下の構造を持つブロックで構成することができ、これを構造化プログラミングと呼びます。

順次:上から下へ向かって記述された順番でプログラムが実行される
　　逐次処理
選択:条件 A なら、処理 1 を行う
　　条件分岐:if 文、if / else 文、if / elif /else 文
反復:条件 A を満たす間、処理 1 を繰り返す
　　繰り返し処理:while 文、for 文

たとえば、以下は構造化されたプログラムの例です。

```
for i in range(1, N + 1):
    if i % 3 == 0:
        print("hoge");
    else:
        x = i
        while (x):
            if x % 10 == 3:
                print("hoge"")
                break
            x //= 10
```

問題

与えられた整数の累積値を計算し、目的の値に達したときの値を求めよ。

入力例	出力例
5 # データの数 120 # 目的の値 50 50 50 10 100	150

INFO1_08_H 構造化2

問題

与えられた整数の列を 0 で区切り、それぞれの部分列の要素数を求めよ。

入力例	出力例
6 # データの数 1 2 0 3 0 8	2 1

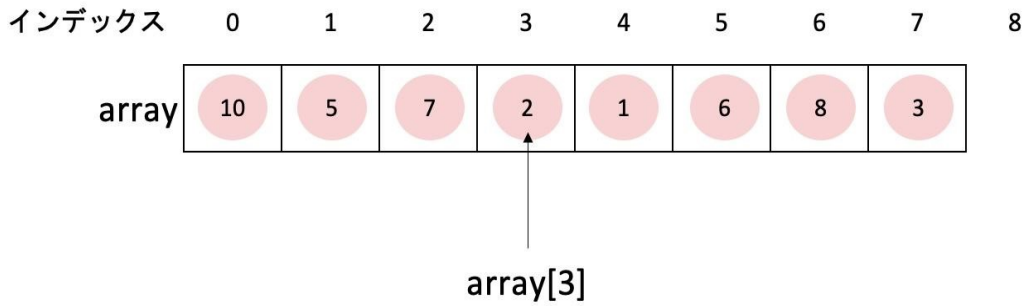
9 配列

条件によって処理を何度も繰り返す繰返し処理によって、コンピュータの強力な計算能力を利用することができました。さらに、大量のデータをメモリに保持しつつ、それらに対して繰り返し処理で読み書きできるようになれば、計算機の計算力と記憶力の双方を活かしてさらに汎用的な情報処理が行えるようになります。このトピックを修了すれば、データの列やリストを変数(メモリ)として扱い、その要素へのアクセスや変更ができるようになります。

9.1 配列

複数の要素を含むデータの列を1つの変数として管理する仕組みを配列と呼びます。基本的な変数には1つの数値しか記録することができませんでしたが、配列は複数のメモリ領域を1つの変数と対応させます。一般的には、1つの配列には同じデータ型の要素が格納されます。

配列に格納される各数値のことを配列の要素と呼び、その要素の数を配列のサイズや長さと呼びます。各要素にはインデックスと呼ばれる通し番号が割り振られます。インデックスは配列の先頭の要素を 0 とする連番で割り振られます。たとえば、以下の図はサイズが 8 の配列を表します。array[3]と参照することで、4 番目の要素である 2 を取得することができます。



さらに、通常の変数と同様に、代入演算の左辺に配列変数名とインデックスを指定すれば、右辺の式の値を書き込むことができます。たとえば、以下のプログラムは配列 array の 7 番目の要素として 8 を代入します。

プログラム
`array[6] = 8`

9.2 リスト

Python では、配列は「リスト」によって実現します。リストは、複数の要素から構成され、要素が順番に並んでいるデータ構造の一種です。一般的な配列との違いは、リストはオブジェクトとして扱われ、異なるデータ型の要素を含めることができます。また、Python は標準でリストをサポートしており、リストに対する様々な操作が標準で利用することができます。

リスト型のオブジェクトは次のように作成します。

[要素1, 要素2, ...]

作成されたリストオブジェクトは変数に代入することができます。

変数名 = [要素1, 要素2, ...]

たとえば、リストの初期化は以下のように記述します。

プログラム

```
lst = []            # 空のリストオブジェクトを変数 lst に代入する
lst = [18, 35, 27] # オブジェクト[18, 35, 27]を変数 lst に代入する
lst = [0]*10        # すべての要素が 0 であるサイズ 10 のリスト list を生成する
```

以下のように、リストは配列と同様の方法で、要素の読み込みと書き込みを行うことができます。

プログラム

```
array = [10, 20, 30]
print(array[0]) # 10
print(array[1]) # 20
print(array[2]) # 30
array[1] = 40
print(array[1]) # 40
```

出力

```
10
20
30
40
```

リストに含まれる要素の数、つまりリストの長さは len 関数で取得できます。たとえば、len 関数は以下のように使います。

プログラム

```
array = [1, 2, 4, 8]
size = len(array)
print(size)  # 4
```

出力

```
4
```

リストの要素は、append メソッドや insert メソッドによって動的に変更することができます。ここで、オブジェクトを表す変数から直接呼び出す関数をメソッドと言います。append メソッドでは引数で与えられたデータをリストの末尾に追加します。たとえば、append メソッドは以下のように使います。

プログラム

```
array = [1, 2, 4]
print(array) # [1, 2, 4]
array.append(8)
print(array) # [1, 2, 4, 8]
```

出力

```
[1, 2, 4]
[1, 2, 4, 8]
```

insert メソッドには2つの引数があり、1 つ目の引数でデータを挿入する場所(インデックス)i、2 つ目の引数に追加する新しいデータ x を指定し、リストの i 番目の要素として x を挿入することができます。たとえば、insert メソッドは以下のように使います。

プログラム

```
array = [1, 2, 4, 8, 16]
array.insert(1, 32)
print(array)  # [1, 32, 2, 4, 8, 16]
```

出力

```
[1, 32, 2, 4, 8, 16]
```

リストの要素を順番に取得するにはいくつかの方法があります。range 関数でリストのインデックス i を順番に生成して、i 番目の要素にアクセスすることができます。たとえば、次のプログラムはリストの要素を順番に出力します。

プログラム

```
array = [32, 64, 128, 256]
for i in range(len(array)):
    print(array[i])
```

出力

```
32
64
128
256
```

for 文を用いる場合は、in 演算にリストを指定することで、その要素を順番に変数に代入しながら処理を実行することができます。たとえば、次のプログラムはリストの要素を順番に出力します。

プログラム

```
array = [32, 64, 128, 256]
for v in array:
    print(v)
```

9.3 演習問題

INFO1_09_A 1次元配列：要素を読む

問題

与えられた数列Aのk番目の要素を出力せよ。

入力例	出力例
5 # 数列の要素数 1 9 7 9 7 2 # k	7

INFO1_09_B 1次元配列：連続要素を読む

前解説

ループを用いれば、配列の要素に順番にアクセスすることができます。次のプログラムは、要素数が n である配列 a の全ての要素を順番に出力します。

プログラム

```
for i in range(0, n):
    print(a[i])
```

リストの部分列の要素は、その範囲のインデックスを順番に指定すれば取得することができます。たとえば次のプログラムは、リスト a の l 番目の要素から (r-1) 番目の要素を順番に出力します。

プログラム

```
a = [1, 2, 4, 8, 16, 32, 64, 128]
l = 2
r = 6
for i in range(l, r):
    print(a[i])
```

出力

```
4
8
16
32
```

Python では、リストのスライス機能によって、リストの部分列を生成することができます。スライス機能は、範囲の開始位置 begin と終了位置 end を [begin:end] のように指定することで、begin 番目から end-1 番目の要素からなるリストを生成します。たとえば次のプログラムは、スライス機能を用いてリスト a の l 番目の要素から (r-1) 番目の要素を順番に出力します。

プログラム

```
a = [1, 2, 4, 8, 16, 32, 64, 128]
l = 2
r = 6
for v in a[l:r]:
    print(v)
```

出力

```
4
8
16
32
```

リストのスライスは、以下のような様々な機能を提供します。

- 部分列を取得する要素の間隔(ステップ)を指定する
- 逆順に部分列を取得する
- スライスで取得した部分列に値を代入する

問題

与えられた数列Aのl番目からr番目の要素を順番に出力せよ。

入力例	出力例
5 # 数列の要素数 1 9 7 9 7 1 # l 3 # r	9 7 9

INFO1_09_C 1次元配列:要素の変更

前解説

変数と同様に、リストの要素は代入演算子 = で何度でも変更(上書き)できます。
次のプログラムは、リストの要素を変更します。

プログラム

`a[0] = 1`
`a[1] = 2`
`a[2] = 3`
`a[2] = a[0] + a[1] + a[2]`
`print(a[2]) # 6`

出力
`6`

問題

全ての要素が 0 である長さがNの数列$A = \{a_0, a_1, \ldots, a_{N-1}\}$に対して、Q回の操作を行う。各操作では、与えられた整数kに対してa_kに 1 を加算する。全ての操作が終了した後の、a_iを順番に出力せよ。

入力例	出力例
5 # 数列の要素数 N 4 # Q 1 2 3 2	0 1 2 1 0

INFO1_09_D 1次元配列:区間要素の変更

前解説

次のプログラムは、インデックスで指定された要素を変更します。

プログラム
`for i in range(10):` `  a[i] = i + 1`
`for i in range(10):` `  print(a[i]) # 1 2 3 4 5 6 7 8 9 10`

問題

全ての要素が 0 である長さがNの数列$A = \{a_0, a_1, \ldots, a_{N-1}\}$に対して、Q回の操作を行う。各操作では、与えられた2つの整数lとrに対して$a_l, a_{l+1}, a_{l+2}, a_r$のそれぞれに 1 を加算する。全ての操作が終了した後の、a_iを順番に出力せよ。

入力例	出力例
5 # 数列の要素数 N 2 # Q 1 # l 3 # r 3 # l 4 # r	0 1 1 2 1

INFO1_09_E 1次元配列：リスト

前解説

一次元配列の用途は主に、リストとバケットに分けられます。

リストは、順番に添え字が付けられたデータの列を管理します。一般的なアプリケーションでは、複数のリストを扱うことになるでしょう。対象のリストを配列名で指定し、そのリストの要素を添え字で指定してアクセスします。

一方、バケットは、いわゆるチェックリストのようなもので、i 番目の要素に i 番目の要素の状態を記録します。例えば、状態としてその番号に対応する要素の有無や個数等を記録します。

ここでは、リストの例を示します。次のプログラムは、2つの配列を2つのリストとして管理します。

プログラム

```
A = [0]*5 # 1 つ目のリスト
B = [0]*5 # 2 つ目のリスト

for i in range(0, 5):
    A[i] = i + 1 # 1 2 3 4 5
for i in range(0, 5):
    B[i] = i + 6 # 6 7 8 9 10

for i in range(0, 5):
    print(A[i], B[i])
```

出力

```
1 6
```

```
2 7
3 8
4 9
5 10
```

問題

$2 \times N$個の整数x_i $(i = 0, 1, \ldots, N-1)$が与えられる。0未満の整数x_iをリストAに、0以上の整数x_iをリストBに順番に格納せよ。

入力例	出力例
4 # N 5 8 -1 7 -3 1 -9 -4	-1 5 -3 8 -9 7 -4 1

INFO1_09_F 1次元配列：バケット

前解説

ここでは、バケットの例を示します。次のプログラムは、1つの配列(リスト)をバケット(チェックリスト)として管理します。

プログラム

```
check = [False]*10 # 初期値が False であるサイズ 10 のチェックリスト

check[7] = True     # 7 番目の要素をチェック
check[1] = True     # 1 番目の要素をチェック
check[0] = True     # 0 番目の要素をチェック

for i in range(10):
    if check[i]:
        print(i)
```

出力

```
0
1
7
```

問題

N個の整数x_i $(i = 0, 1, \ldots, N-1)$が与えられる。与えられた整数をすべて、重複無しに出力せよ。

入力例	出力例
6 # N 1 1 3 1 2 3	1 2 3

10　文字・文字列1

コンピュータで扱うデータには、数値やテキスト(文字列)があります。プログラムは、数値に対する演算だけでなく、文字列の操作も行います。数値も文字列もデータベースに保存されデータ解析の対象となります。このトピックを修了すれば、コンピュータにおける文字の内部表現を理解し、文字や文字列を扱えるようになります。

10.1文字

アプリケーションの利用者側から見える「文字」は、人間が認識できるアルファベット、ひらがな、数字など様々です。一方、これらはコンピュータの内部では、数値として扱われています。つまり、コンピュータは、整数値と文字を対応させて文字情報を処理しています。各文字には、アスキーコードと呼ばれる整数が割り当てられています。アスキーコードの対応表の内容を全て覚える必要はありませんが、文字は体系的な分類に基づいて数値として扱われているということを認識しておく必要があります。

以下に、文字とアスキーコードの対応表を示します。

文字	コード
空白	32
!	33
"	34
#	35
$	36
%	37
&	38
'	39
(	40
)	41
*	42
+	43
,	44
-	45
.	46
/	47

文字	コード
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
:	58
;	59
<	60
=	61
>	62
?	63

文字	コード
@	64
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79

文字	コード
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
[	91
¥	92
]	93
^	94
_	95

文字	コード
`	96
a	97
b	98
c	99
d	100
e	101
f	102
g	103
h	104
i	105
j	106
k	107
l	108
m	109
n	110
o	111

文字	コード
p	112
q	113
r	114
s	115
t	116
u	117
v	118
w	119
x	120
y	121
z	122
{	123
|	124
}	125
~	126
DEL	127

10.2　文字列

文字列は文字の列で構成されます。プログラミング言語によって文字列を扱う方法は様々ですが、一般的に以下のような操作を行うことができます。

- 文字列の特定の位置の文字を参照・変更する
- 文字列の長さ（文字の数）を取得する
- 文字列を数値化する
- 文字列について様々な操作を行う（2つの文字列を連結する、部分文字列を取り出す、部分文字列を別な文字列に置換するなど）

10.3 演習問題

INFO1_10_A 文字

問題

標準入力から1つの文字cが1行に与えられる。cを出力せよ。

入力例	出力例
a	a
z	z

INFO1_10_B 文字 to アスキーコード

前解説

ord 関数で、文字から対応するアスキーコードを取得することができます。たとえば、以下のプログラムは ord 関数を使って文字のアスキーコードを出力します。

プログラム
`print(ord("A"))` `print(ord("0"))`
出力
65 48

ord 関数の引数は文字列ですが、1つの文字のアスキーコードを取得するために、その長さは1である必要があります。

問題

標準入力から1つの文字cが1行に与えられる。cの ASCII コード(アスキーコード)を出力せよ。

入力例	出力例
a	97
3	51

INFO1_10_C アスキーコード to 文字

前解説

chr 関数で、アスキーコードから対応する文字を取得することができます。たとえば、以下のプログラムは chr 関数を使ってアスキーコードが表す文字を出力します。

プログラム
`print(chr(66))` `print(chr(49))`
出力
`B` `1`

問題

標準入力からアスキーコードを表す1つの整数cが1行に与えられる。cに対応する文字を出力せよ。

入力例	出力例
97	a
51	3

INFO1_10_D 文字の種類

前解説

文字列がどのような文字の種類で構成されるかは、文字列型から使える様々なメソッドにより判定することができます。たとえば、以下のメソッドが利用できます。

メソッド	機能
isdigit	1 文字以上の文字列ですべての文字が数字であれば True
isalpha	1 文字以上の文字列ですべての文字がアルファベットであれば True
islower	1 文字以上の文字列ですべての文字が英小文字であれば True
isupper	1 文字以上の文字列ですべての文字が英大文字であれば True

たとえば、以下のプログラムは文字列の種類を判定します。

プログラム

`"123".isdigit()  # True`
`"A4".isdigit()  # False`
`"abab".isalpha() # True`
`"abab".islower() # False`
`"abab".isupper() # True`
`"XYZ".isupper()  # False`

出力

`True`
`False`
`True`
`True`
`False`
`True`

問題

標準入力から1つの文字cが1行に与えられる。cが数字のとき"digit"、英小文字のとき"lower"、英大文字のとき"upper"、これら以外のとき"other"と出力せよ。

入力例	出力例
a	lower
A	upper
3	digit

INFO1_10_E 文字列

問題

標準入力から1つの文字列sが1行に与えられる。sを出力せよ。

入力例	出力例
hello	hello
aoj	aoj

INFO1_10_F 文字列の長さ

前解説

文字列の長さは len 関数で取得することができます。たとえば、次のプログラムは文字列の長さを出力します。

プログラム

```
len(“abab”)    # 4
len(“”)        # 0
len(“12345”)   # 5
```

出力

```
4
0
5
```

問題

標準入力から1つの文字列sが1行に与えられる。sの長さを出力せよ。

入力例	出力例
hello	5
aoj	3

INFO1_10_G 文字列の文字

前解説

文字列の i 文字目の文字は、リストと同様にインデックスを指定して取得することができます。たとえば、次のプログラムは文字列の指定された文字を出力します。

プログラム

```
problems = "ABCDEF"
print(problems[0]) # A
print(problems[3]) # D
print(problems[5]) # F
```

出力

```
A
D
F
```

問題

標準入力から1つの文字列Sが1行に与えられる。Sに含まれる文字を順番にそれぞれ1行に出力せよ。

入力例	出力例
hello	h e l l o

INFO1_10_H 文字列の数値化

問題

標準入力から1つの文字列Sが1行に与えられる。Sが整数を表す場合、$S+1$を出力し、それ以外の場合は文字列Sを出力せよ。

入力例	出力例
abc	abc
99	100

11 入力2

プログラミング言語やそれらの機能によって、データを入力する方法は様々です。また、入力するデータの形状も様々です。このトピックでは、行単位で与えられた「空白を含む文字列」を入力し、その文字列を目的に応じて扱う方法を学びます。このトピックを修了すれば、行単位で与えられた文字列をリスト化したり、その要素を数値化することができるようになります。

11.1 行入力

Python の input 関数は、1行単位で「空白を含む文字列」を入力します。つまり、1行の中に、空白区切りで文字列や数値が含まれている場合は、これらを空白区切りの文字列(これをトークンと呼びます)の列に変換する必要があります。

Python では、文字列に対する split メソッドによって、指定された文字で文字列を分割することができます。分割された複数の文字列はリストとして取得することができます。split メソッドに引数を指定しない場合は、文字列を空白で区切ります。たとえば、以下のように文字列"nice to meet you"に対して split メソッドを用いると、"nice", "to", "meet", "you"の4つの文字列に分割されます。

プログラム

```
str = "nice to meet you"
tokens = str.split()
print(tokens[0]) # nice
print(tokens[1]) # to
print(tokens[2]) # meet
print(tokens[3]) # you
print(tokens)    # ['nice', 'to', 'meet', 'you']
```

出力

```
nice
to
meet
you
['nice', 'to', 'meet', 'you']
```

1 行に複数の整数が空白区切りで与えられる場合は、その1行の文字列に対して split を行ってから、各要素を個別にキャストし、整数のリストへ変換します。たとえば、以下のプログラムは、1つの文字列で表された複数の整数をリストに変換します。

プログラム

```
str = “1 2 3 4”
tokens = str.split()

for i in range(len(tokens)):
    tokens[i] = int(tokens[i])

print(tokens[0]) # 1
print(tokens[1]) # 2
print(tokens[2]) # 3
print(tokens[3]) # 4
print(tokens)    # [1, 2, 3, 4]
```

出力

```
1
2
3
4
[1, 2, 3, 4]
```

map 関数を用いれば、この処理をより簡潔に記述できます。
たとえば、以下のプログラムは map 関数を使って文字列のリストを整数のリストに変換します。

プログラム

```
str = "1 2 3 4"
tokens = list(map(int, str.split()))

print(tokens[0]) # 1
print(tokens[1]) # 2
print(tokens[2]) # 3
print(tokens[3]) # 4
print(tokens)    # [1, 2, 3, 4]
```

出力

```
1
2
3
4
[1, 2, 3, 4]
```

11.2演習問題

INFO1_11_A 1行空白区切り文字列

問題

空白区切りで1つ以上の文字列が１行に与えられる。ただし、空白は連続で現れる場合がある。与えられた文字列を順番にそれぞれ１行に出力せよ。

入力例	出力例
aizu online judge	aizu online judge

INFO1_11_B 1行空白区切り文字列逆順

問題

空白区切りで1つ以上の文字列が１行に与えられる。ただし、空白は連続で現れる場合がある。与えられた文字列を逆順にそれぞれ１行に出力せよ。

入力例	出力例
aizu online judge	judge online aizu

INFO1_11_C 複数行

問題

１行目に整数Nが与えられる。続いてN行の入力が与えられる。各行は、空白区切りの1つ以上の文字列で構成される。ただし、空白は連続で現れる場合がある。入力の各行について、与えられた文字列を１つの空白区切りで順番に出力せよ。

入力例	出力例
3 # N to advance knowledge for humanity	to advance knowledge for humanity
2 Algorithms and Data Structures	Algorithms and Data Structures

INFO1_11_D 複数行　逆順

問題

１行目に整数Nが与えられる。続いてN行の入力が与えられる。各行は、空白区切りの1つ以上の文字列で構成される。ただし、空白は連続で現れる場合がある。入力の各行について、与えられた文字列を１つの空白区切りで逆順に出力せよ。

入力例	出力例
3 # N to advance knowledge for humanity	to knowledge advance humanity for
2 Algorithms and Data Structures	and Algorithms Structures Data

INFO1_11_E 行の数列変換

問題

整数の数列の要素a_iが空白区切りで１行に与えられる。$a_i + 1$を順番に出力せよ。

入力例	出力例
1 3 5 7 9	2 4 6 8 10

INFO1_11_F 複数行の数列変換

問題

１行目に整数Nが与えられる。続いてN行の入力が与えられる。各行は、空白区切りの1つ以上の整数a_iで構成される。入力のj ($j = 1,2,..N$)行目について、$a_i + j$を順番に出力せよ。

入力例	出力例
3 # N 1 2 3 4 5 6 7 8 9	2 3 4 5 7 8 9 11 12

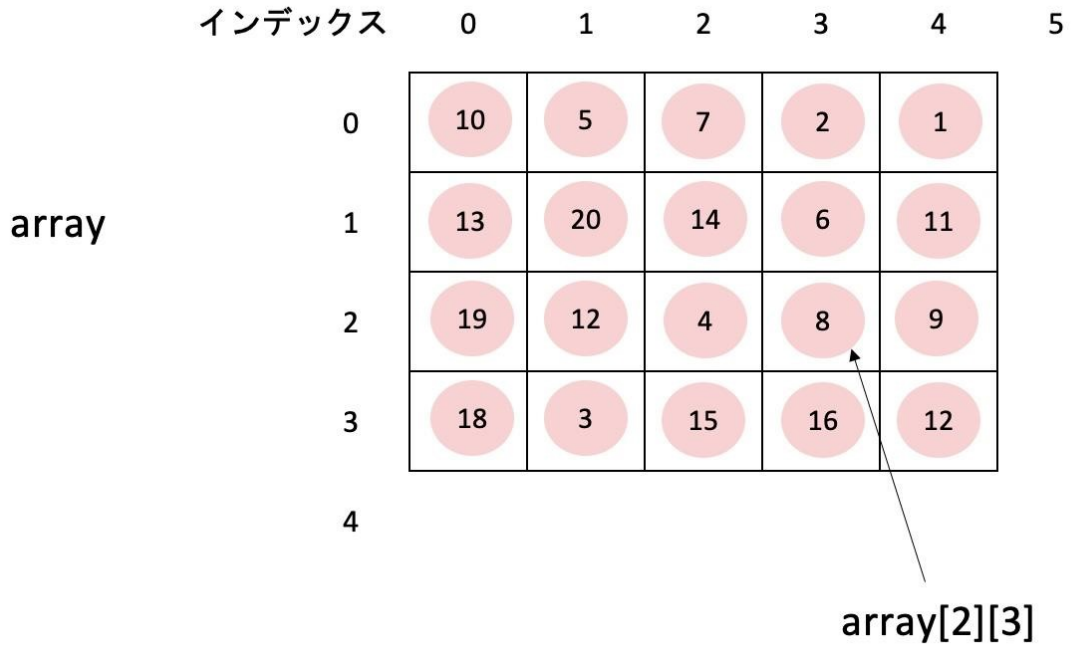
12　配列2

配列やリストは、1次元配列と呼ばれ1方向に連続した要素を格納します。表計算のシートのように、多方向に連続した要素を管理する、いわゆる多次元の配列変数を応用することで、表や空間の中のデータを管理できるようになります。このトピックを修了すれば、多次元の配列を扱えるようになります。

12.1多次元配列

1方向に連続した要素を格納している配列を1次元配列、2方向以上に連続した要素を格納している配列を多次元配列と呼びます。特に、2方向に連続した要素を格納している配列を2次元配列と呼びます。

2次元配列では縦方向と横方向にそれぞれサイズがあり、インデックスが割り振られます。要素を参照するときにはインデックスが2つ必要になり、一般的には1つ目が縦方向のインデックス、2つ目が横方向のインデックスを表します。たとえば、以下の図は縦方向のサイズが 4、横方向のサイズが 5 の配列を表します。縦方向3番目、横方向4番目の要素 array[2][3]を参照すると、8 を取得することができます。



12.2　2次元リスト

Python では、2次元配列を、リストのリスト、つまり2次元リストで表します。リストの各要素がリストを含むデータ構造になります。たとえば、次のプログラムは2次元リストを生成して、その要素を出力します。

プログラム

```
puzzle = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 0]
]
print(puzzle)
print(puzzle[0][0])
print(puzzle[1][2])
print(puzzle[2][0])
print(puzzle[2][2])
```

出力

```
[[1, 2, 3], [4, 5, 6], [7, 8, 0]]
1
6
7
0
```

２次元リストは、内包表記を使って初期化を行います。内包表記により、繰返し処理の結果をリストへ格納する処理をより簡潔に記述することができます。たとえば、以下の2つのプログラムは同じ処理を行い、リストを初期化します。

プログラム

```
even = []
for i in range(5):
  even.append(i*2)
print(even)
```

```
even = [i*2 for i in range(5)]
print(even)
```

出力

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

たとえば、次のプログラムは２次元リストを初期化して、要素を変更するプログラムです。

プログラム

```
grid = [[0 for j in range(5)] for i in range(4)]
grid[1][1] = 1
grid[2][3] = 2
grid[3][4] = 4
print(grid)
# [
#   [0, 0, 0, 0, 0],
#   [0, 1, 0, 0, 0],
#   [0, 0, 0, 2, 0],
#   [0, 0, 0, 0, 4]
# ]
```

出力

```
[[0, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 0, 2, 0], [0, 0, 0, 0, 4]]
```

12.3 演習問題

INFO1_12_A 2次元配列：要素の読み込み

問題

N行M列から成る表Aに対して、指定された要素$a_{i,j}$を出力せよ。

入力例	出力例
3 4 # N M 1 2 3 4 5 6 7 8 9 10 11 12 1 2 # i j	7

INFO1_12_B 2次元配列：区画要素の読み込み

問題

N行M列から成る表Aの、i_1行目からi_2行目、i_3列目からi_4列目に含まれる要素$a_{i,j}$を順番に出力せよ。

入力例	出力例
4 5 # N M 1 2 3 4 5 6 7 8 9 10 11 13 14 15 16 17 18 19 20 21 1 2 # i1 j1 3 3 # i2 j2	8 9 14 15 19 20

INFO1_12_C 2次元配列：要素の更新問題

問題

N行M列から成る表Aがある。初期状態でAの要素$a_{i,j}$は全て0である。Q個の整数の組(i, j)が質問として与えられる。各質問についてAの要素$a_{i,j}$が1ならばそれを0に、0ならば1に変換せよ。

入力例	出力例
4 5 10 # N M Q 0 3 1 4 2 2 1 3 3 2 3 0 1 4 1 3 3 1 0 2	0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0

INFO1_12_D 2次元配列：区画要素の更新

問題

N行M列から成る表Aがある。初期状態でAの要素$a_{i,j}$は全て0である。Q個の質問が与えられる。各質問は2つの整数の組(i_1, j_1)と(i_2, j_2)から成る。各質問についてAのi_1行目からi_2行目、j_1列目からj_2列目の要素$a_{i,j}$に1を加算せよ。

入力例	出力例
4 5 4 # N M Q 0 3 1 4 3 3 3 4 0 2 3 2 2 2 3 3	0 0 1 1 1 0 0 1 1 1 0 0 2 1 0 0 0 2 2 1

INFO1_12_E 3次元配列

問題

$X[m] \times Y[m] \times Z[m]$の３次元空間に$N$個の点が与えられる。続いて$Q$個の質問が与えられる。各質問は3つの整数$x, y, z$から成る。座標$(x, y, z)$に点が存在するかどうかを判定せよ。

入力例	出力例
3 4 5 # X Y Z 5 # Q 0 0 1 1 2 1 3 1 3 0 3 1 2 1 5 3 1 2 3 1 2 1 2 1 5	0 1 1

13　関数

データを受け取り、定められた処理を実行して計算結果を返す仕組みを関数と呼びます。input や print のように、既存の関数を呼び出すことで、汎用的な処理を(自分で定義することなく)実行することができます。一方、関数を定義することができれば、より読みやすいプログラムを書くことができます。このトピックを修了すれば、自作の関数を定義し、その関数を呼び出して使えるようになります。

13.1 関数

プログラミングでは、よく使う汎用的な処理を定義しておき、それを適宜呼び出して実行することができる、関数という仕組みがあります。関数は主に以下の2つに分類されます。

- 組み込み関数は、プログラミング言語に組み込まれている(標準で実装されている)関数で、その言語の開発者たちによって実装された汎用的な処理を提供します。たとえば、input や print は Python の組み込み関数です。
- 自作関数は、プログラマ自身が実装する関数です。

数学には、たとえば$y = f(x)$のように、ある値xが決まるとyが決まる関数という概念があります。プログラムの関数は、データを受け取り、そのデータを元になんらかの処理を行い、計算結果を返します。関数に渡し値を引数と呼びます。関数を実行して得られる結果を戻り値と呼びます。

引数が存在しない関数や、戻り値が存在しない関数もあります。いつくかのプログラミング言語では、戻り値のない関数をプロシージャと呼びます。

13.2　関数の定義

自作の関数は以下のように定義します。

```
def 関数名(引数1, 引数2, ...):
    処理
    return 戻り値
```

関数の定義は、def というキーワードで始まり、関数名の後の( )の中に引数を定義します。引数は並べて複数指定することができます。

関数の処理は、インデントを1つ入れて記述します。戻り値があれば、return 文で処理の結果を返します。

13.3 関数の呼び出し

定義された関数は、関数名と( )に記述した引数を指定して、プログラムの処理や数式の中で呼び出すことができます。

たとえば、次のプログラムは、数列の要素の合計を関数によって求めます。

プログラム

```
def sum_array(A):
    sum = 0
    for a in A:
        sum += a
    return sum

A = [18, 22, 5, 61, 10, 34] # 処理開始
result = sum_array(A)
print(result) # 150
```

出力

```
150
```

13.4 演習問題

INFO1_13_A 関数

問題

以下の処理を行う関数を作成せよ。

- 1つの整数aを引数として受け取る
- aに１を加算した値を返す

入力として1つの整数xが与えられる。上記の関数にxを渡して得られた値を出力せよ。

入力例	出力例
3	4

INFO1_13_B 複数の引数

問題

以下の処理を行う関数を作成せよ。

- 1つの整数nと1つの文字cを引数として受け取る
- 文字cのみで構成された長さnの文字列を出力して改行する
- 値を返さない

入力として、整数Nと文字Cの組が与えられる。 上記の関数にCとNを渡して実行した結果を出力せよ。

入力例	出力例
3 A	aaa

INFO1_13_C 配列の引数

問題

以下の処理を行う関数を作成せよ。

- 整数の配列Aを引数として受け取る
- Aの要素を１つ返す

入力として、1つの数列Xが与えられる。上記の関数にXを渡して得られた値を出力せよ。

入力例	出力例
3 1 2 3	1

INFO1_13_D 配列の引数と処理

問題

以下の処理を行う関数を作成せよ。

- 整数の配列Aを引数として受け取る
- 以下の形式でAの要素を１行に、カンマ区切りで出力する
  - $(a_0, a_1, \ldots, a_{N-1})$
  - 数列を( )で囲み、カンマの後に1つの空白を入れることに注意せよ。
- 値を返さない

1つの数列Xが与えられる。上記の関数にXを渡して実行した結果を出力せよ。

入力例	出力例
3 1 2 3	(1, 2, 3)

INFO1_13_E 配列を返す関数

問題

以下の処理を行う関数を作成せよ。

- 1つの整数nを引数として受け取る
- 長さnの配列Aを生成する
- Aのi番目の要素$a_i(i = 0, 1, \ldots, n-1)$を$i$で初期化する
- 配列Aを返す
-

入力として、整数Nが与えられる。上記の関数にNを渡して得られた数列を出力せよ。出力には、INFO1_13_Dで作成した関数を利用する。

入力例	出力例
3	(0, 1, 2)

INFO1_13_F 配列を更新する関数

問題

以下の処理を行う関数を作成せよ。

- 整数の配列Aと整数cを引数として受け取る
- Aの要素$a_i(i = 0, 1, \ldots, N-1)$それぞれに$c$を掛けた値を要素$b_i$とする数列$B$を生成して返す

入力として、1つの数列Xと整数kが与えられる。数列Xの要素と、上記の関数にXとkを渡して得られた数列の要素を出力せよ。出力には、INFO1_13_Dで作成した関数を利用する。

入力例	出力例
3 1 2 3 2	(1, 2, 3) (2, 4, 6)

INFO1_13_G 配列から配列を生成する関数

問題

以下の処理を行う関数を作成せよ。

- 2つの配列A, Bを引数として受け取る
- Aの要素$a_i (i = 0, 1, \ldots, n-1)$に$B$の要素$b_i$を加算した値$c_i$を要素とする数列$C$を生成して返す

入力として数列X, Yが与えられる。 Xの要素、Yの要素、上記関数にXとYを渡して得られた数列の要素を空白区切りで出力せよ。出力には、INFO1_13_D で作成した関数を利用する。

入力例	出力例
3 1 2 3 4 5 6	(1, 2, 3) (4, 5, 6) (5, 7, 9)

14　探索

大量のデータの中から、欲しい情報を取り出す処理は、情報処理の基本と言えます。データの特徴を考慮した探索アルゴリズム(方法)を使うことが重要です。このトピックを終了すれば、配列に格納されたデータから、特定のデータを探すための効率的な方法を理解し、それらのプログラムを書けるようになります。

14.1 探索

データの列から特定のデータを探し出す操作を探索と呼びます。実際のアプリケーションでは、探索の対象となるデータは、様々な形で格納されています。ここでは、1次元の配列に格納された整数のデータを探す問題を考えます。

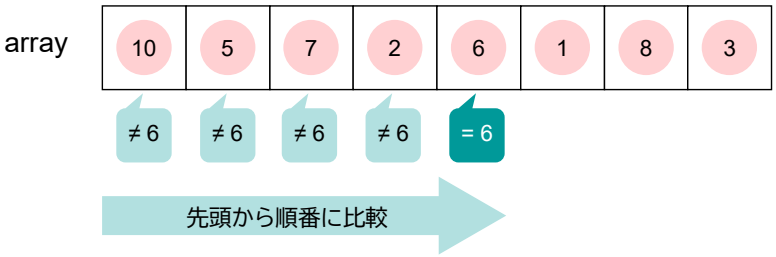
探索の処理には、主に以下の2つのデータを扱います

- 探索の対象となる配列データ
- 探したいデータ（一般的には、質問としていくつか与えられる）

データの列から指定したデータを代表的な探すアルゴリズムとして、線形探索と二分探索が知られています。

14.2 線形探索

線形探索は、配列の先頭から要素を順番に見ていき、データが見つかった時点で探索を終了します。たとえば、リスト array = [10, 5, 7, 2, 6, 1, 8, 3]から 6 を線形探索で探索する場合、以下のような動作となります。



次のプログラムは、リスト A から指定されたデータ key を線形探索で探します。A に key が含まれていれば True を、含まれていなければ False を返します。

プログラム

```
def linearSearch(A, key):
    for a in A:
        if a == key:
            return True
    return False
```

```
X = [10, 5, 7, 2, 6, 1, 8, 3]
print(linearSearch(X, 6))
print(linearSearch(X, 3))
print(linearSearch(X, 4))
```

出力

```
True
True
False
```

Python では、in 演算子で線形探索と同様の処理を行うことができます。in 演算子はリストなどに特定の要素が含まれているかを判定し、その結果を True または False で返します。

次のプログラムは、上記のものと同様に、リスト A から指定されたデータ key を線形探索で探します。A に key が含まれていれば True を、含まれていなければ False を返します。

プログラム

```
def linear_search(A, key):
    return key in A
```

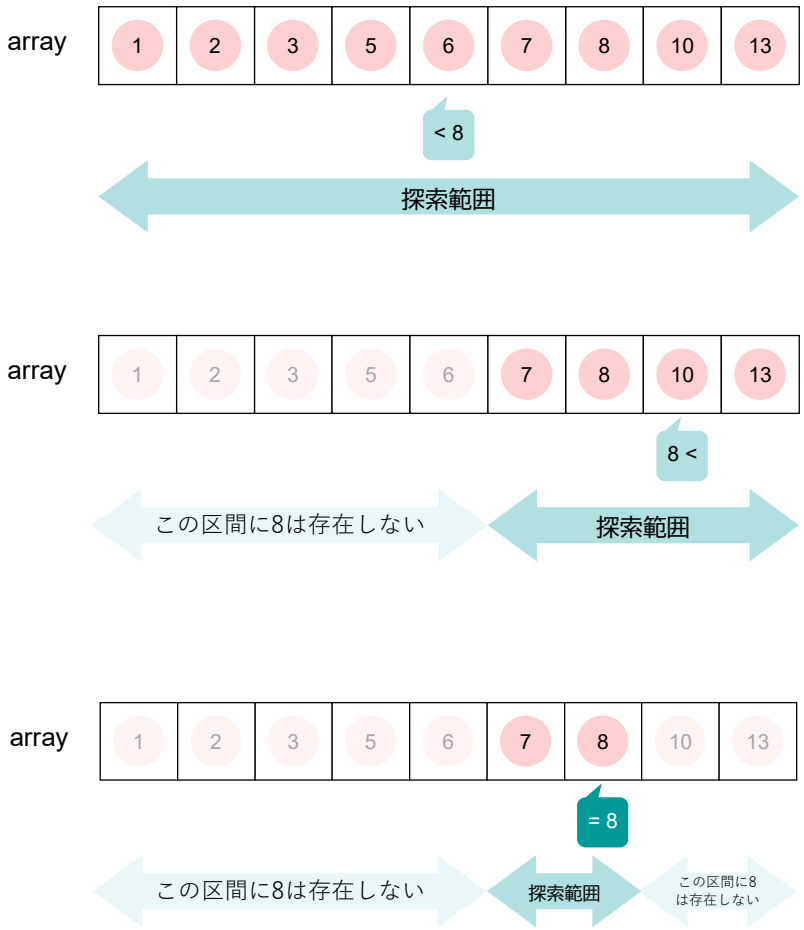
14.3　二分探索

コンピュータで扱うデータは、多くの場合、整理整頓されています。なぜなら、整理されたデータは探しやすいからです。つまり、データが整理されていれば、より効率的な探索アルゴリズムを適用することができます。

二分探索は昇順に整列されたリストに対して、以下の手順で探索をします。

1. リストの中央の値を参照する
2. 探索する値が中央の値と一致した場合、探索終了（→探索する値がリストに存在する）
3. 探索する値が中央の値よりも大きい場合、探索するリストの範囲を後半のみにする
4. 探索する値が中央の値よりも小さい場合、探索するリストの範囲を前半のみにする
5. 探索するリストの範囲が１未満になった場合、探索終了（→探索する値がリストに存在しない）
6. １に戻る

たとえば、リスト array = [1, 2, 3, 5, 6, 7, 8, 10, 13]から 8 を二分探索で探索する場合、以下のような動作となります。



次のプログラムは、リスト A から指定されたデータ key を線形探索で探します。A に key が含まれていれば True を、含まれていなければ False を返します。

プログラム

```
def binarySearch(A, key):
    l = 0
    r = len(A)
    while l < r:
        m = (l + r) // 2
        if A[m] == key:
            return True
        if A[m] < key:
            l = m + 1
        else:
            r = m
    return False

X = [1, 2, 2, 4, 5, 8, 9, 9, 13, 14, 14, 18]
print(binarySearch(X, 9))
print(linearSearch(X, 14))
print(linearSearch(X, 6))
```

出力

```
True
True
False
```

14.4　線形探索と二分探索の比較

データを探索するときには、そのデータのサイズや特徴を考慮して、適切なアルゴリズムを選択する必要があります。次の表は、線形探索と二分探索の特徴を示します。

	良い点	悪い点
線形探索	どのようなデータの並びでも、目的のデータを探すことができる。	比較回数が多く、計算効率が悪い
二分探索	比較回数が少なく、計算効率が良い	データが昇順に並んでいる必要がある(ただし、多くの場合は整列されているため、問題にならない)

次の表は、線形探索と二分探索の計算効率を比較します。

データの数データの数	線形探索の比較回数	二分探索の比較回数
10	10	3
100	100	7
1000	1000	10
10000	10000	13
100000	100000	17
1000000	1000000	20
…	…	…
1000000000	1000000000	30

データの数をNとすると、線形探索の比較回数はNに比例し、二分探索の比較回数は$\log_2 N$ に比例します。

14.5 演習問題

INFO1_14_A 探索

問題

数列Aと整数keyが与えられる。Aにkeyが含まれるかどうか判定せよ。

入力例	出力例
6 # 数列 A の要素数 N 8 1 2 2 3 5 # 数列 A の要素 3 # key	Yes
6 8 1 2 2 3 5 4	No

INFO1_14_B 要素の位置

問題

数列Aと整数keyが与えられる。以下の疑似コードに基づき線形探索を行った場合、keyが最初に現れる位置を出力せよ。ただし、Aの最初の要素の位置を 0 とする。

```
linearSearch(A, key):
    N = length of A
    for i = 0 to N - 1:
        if a == key:
            return i
    return -1
```

入力例	出力例
6 8 1 2 4 4 5 4	3
6 8 1 2 4 4 5 3	-1

INFO1_14_C クエリによる探索

問題

数列Aといくつかの質問が与えられる。各質問で与えられるkeyについて、Aにkeyが含まれるかどうか判定せよ。

入力例	出力例
6 # 数列 A の要素数 N 8 1 2 2 3 5 # 数列 A の要素 3 # 質問の数 2 # 1つ目の質問 6 # 2つ目の質問 5 # 3つ目の質問	Yes No Yes

INFO1_14_D 二分探索

問題

数列Aと整数keyに対して、以下の疑似コードに基づく二分探索を実行せよ。

```
binarySearch(A, key):
    l = 0
    r = length of A
    while l < r:
        m = (l + r) / 2        # 切り捨て除算
        if A[m] == key:
            break
        if A[m] < key:
            l = m + 1
        else:
            r = m
```

入力例	出力例
8 1 2 3 4 5 6 7 8 2 8 1 2 3 4 5 6 7 8 6	[0, 8) [0, 4) [0, 2) [0, 8) [5, 8) [5, 6)

INFO1_14_E クエリによる二分探索

問題

数列Aといくつかの質問が与えられる。各質問で与えられるkeyについて、Aにkeyが含まれるかどうか判定せよ。

入力例	出力例
8 1 2 3 4 5 6 7 8 5 4 7 9 8 0	Yes Yes No Yes No

INFO1_14_F 線形探索と二分探索の比較

問題

数列Aと整数keyが与えられる。以下の疑似コードに基づいた線形探索と二分探索を比較せよ。(疑似コードは14_B, 14_D を参照)

入力例	出力例
8 1 2 3 4 5 6 7 8 2	1 2 # 線形探索の結果 5 3 2 # 二分探索の結果
8 1 2 3 4 5 6 7 8 6	1 2 3 4 5 6 5 7 6

INFO1_14_G 線形探索と二分探索の比較回数

問題

要素が昇順に並んだ要素数Nの数列Aを作る。数列Aに対して、以下の疑似コードに基づく線形探索と二分探索を行う。それぞれについて、keyと比較する回数が最も多い場合の比較回数を求めよ。（疑似コードは 14_B, 14_D を参照）

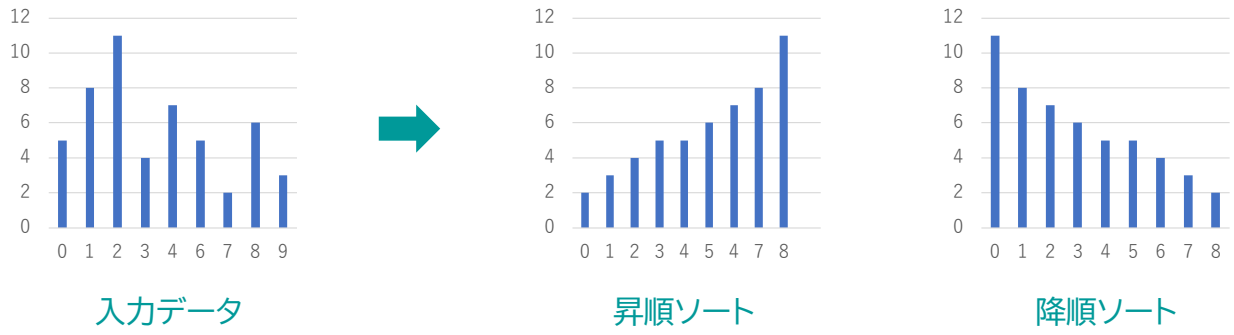
入力例	出力例
8 16 262144	8 4 16 5 19

15　整列

物を整理して探しやすくすることは日常生活においても良き習慣です。情報処理では、整列されたデータは効率的に探索しやすいことを学びました。このトピックを修了すれば、配列のデータを整列できるようになります。

15.1 ソート

配列の要素を昇順、または降順に並び替える処理をソートまたは整列と呼びます。



昇順に整列された配列$A = a_0, a_1, \ldots, a_{N-1}$は、以下の大小関係を満たします。

$$a_0 \leq a_1 \leq \cdots \leq a_{N-1}$$

これを単調増加と呼びます。

一方、降順に整列された配列$A = a_0, a_1, \ldots, a_{N-1}$は、以下の大小関係を満たします。

$$a_0 \geq a_1 \geq \cdots \geq a_{N-1}$$

これを単調減少と呼びます。

15.2 ソートのアルゴリズム

配列の要素をソートするために、様々なアルゴリズムが考案されています。次の表は、代表的なソートアルゴリズムです。

ソートのアルゴリズム	特徴	計算効率
バブルソート	隣り合う要素を繰り返し交換していく	悪い・非実用的
選択ソート	最小値を探索して交換していく	悪い・非実用的
挿入ソート	昇順の列に要素を挿入していく	悪い・非実用的
マージソート	高度なプログラミングテクニックを用いる	良い・実用的
クイックソート	高度なプログラミングテクニックを用いる	良い・実用的
ヒープソート	高度なデータ構造を用いる	良い・実用的
…		

※これらのアルゴリズムはAOJのコースALDS1で学ぶことができます

バブルソート

バブルソートとは、隣接する要素の大小を比較しながら整列させるソート

1. 以下の手順で、先頭から小さい順にソート済みの要素を確定していく
    - 配列の後方から前方に向かって、隣り合う要素を比較し、順番が逆なら交換する処理を、ソート済みの要素の手前まで繰り返す

プログラム

```
def bubbleSort(A):
    for i in range(len(A)-1):              # 先頭から順番に決定する
        for j in range(len(A)-1, i, -1):   # 後方から交換していく
            if A[j-1] > A[j]:
                A[j-1], A[j] = A[j], A[j-1]
```

選択ソート

選択ソートは、リストから最小値を探し、先頭要素と入れ替えながら整列する

1. 以下の手順で、先頭から小さい順にソート済みの要素を確定していく
    - 最小値を探索する
    - 最小値と未ソートの先頭の要素を入れ替える

プログラム

```
def selectionSort(A):
    n = len(A)
    for i in range(n):              # 先頭から順番に決定する
        minj = i
        for j in range(i + 1, n): # 最小値の位置を探索する
            if A[j] < A[minj]:
                minj = j
     A[i], A[minj] = A[minj], A[i]
```

挿入ソート

挿入ソートは、未整列のリストからデータを1つ取り出し、整列済みのリストの適切な位置にデータを挿入することを繰り返し、整列する

1. 先頭の要素をソート済みにする
2. 次の要素を整列済みのリストの適切な位置に挿入する
3. 要素が無くなるまで２を繰り返す

プログラム

```
def insertionSort(A):
    for i in range(1, len(A)):      # 2番目の要素から順番に選んでいく
        v = A[i]
        j = i - 1
        while j >= 0 and A[j] > v:  # 挿入する位置を探しながら後方へずらす
            A[j + 1] = A[j]
            j -= 1
        A[j + 1] = v
```

15.3 組み込み関数・メソッド

Python では、sort メソッドや sorted 関数でリストの要素をソートすることができます。
次のプログラムは、sort メソッドでリストをソートし、リストの中身を変更します。

プログラム

```
array = [5, 7, 2, 3, 1]
array.sort()
print(array)
```

出力

```
[1, 2, 3, 5, 7]
```

sorted 関数は、引数で渡したリストをソートし、ソートされたリストを返します(元のリストは変更されない)。
次のプログラムは、sort 関数でリストをソートし、リストの中身を変更します。

プログラム

```
array = [5, 7, 2, 3, 1]
array_sorted = sorted(array)
print(array_sorted)
```

出力

```
[1, 2, 3, 5, 7]
```

sorted 関数は、引数 sort メソッドの第 1 引数、sorted 関数の第 2 引数に reverse = True を設定することでリストを降順にソートすることができます。
次のプログラムは、sort メソッドでリストを降順にソートします。

プログラム

```
array = [5, 7, 2, 3, 1]
array_sorted = sorted(array, reverse = True)
array.sort(reverse = True)
print(array)
print(array_sorted)
```

出力

```
[7, 5, 3, 2, 1]
[7, 5, 3, 2, 1]
```

15.4 演習問題

INFO1_15_A 昇順の判定

問題

与えられた数列Aの要素が昇順に並べられているか判定せよ。

入力例	出力例
5 1 2 4 7 9 7 1 2 7 5 8 10 11	Yes No

INFO1_15_B 要素の交換

問題

与えられた数列Aに対して、指定された2つの要素を交換せよ。

入力例	出力例
5 1 2 4 7 9 1 3	1 7 4 2 9

INFO1_15_C 要素の反転

問題

与えられた数列Aに対して、指定された区間を逆順にせよ。

入力例	出力例
5 1 2 4 7 9 1 3	1 7 4 2 9

INFO1_15_D 要素の回転

問題

与えられた数列Aと整数kに対して、要素a_kが先頭になるように回転せよ。ここで、回転とは数列の全ての要素を一様に前方(または後方)へ移動し、移動先が数列の範囲を超えた場合はその要素を数列の末尾(または先頭)に移動する操作である。

入力例	出力例
5 1 2 4 7 9 2	4 7 9 1 2
5 1 2 4 7 9 1	2 4 7 9 1

INFO1_15_E 昇順に整列

問題

与えられた数列Aを昇順に出力せよ。

入力例	出力例
5 5 1 3 2 4	1 2 3 4 5

INFO1_15_F 降順に整列

問題

与えられた数列Aを降順に出力せよ。

入力例	出力例
5 5 1 3 2 4	5 4 3 2 1

INFO1_15_G 整列の状態

問題

与えられた数列Aが以下のどの条件を満たすかを判定せよ。

狭義単調増加　$a_0 < a_1 < \cdots < a_{N-1}$
単調増加　$a_0 \leq a_1 \leq \cdots \leq a_{N-1}$
狭義単調減少　$a_0 > a_1 > \cdots > a_{N-1}$
単調減少　$a_0 \geq a_1 \geq \cdots \geq a_{N-1}$

入力例	出力例
4 1 2 3 4	2　# 狭義単調増加
4 1 2 2 4	1　# 単調増加
4 1 1 1 1	0　# 単調増加かつ単調減少
4 4 2 2 1	-1　# 単調減少
4 4 3 2 1	-2　# 狭義単調減少

模範解答

INFO1_01_A

```
print(89)
```

INFO1_01_B

```
print("aizu")
```

INFO1_01_C

```
print(1, 2)
```

INFO1_01_D

```
print(10)
```

INFO1_01_E

```
print(0)
print(1)
print(100)
```

INFO1_02_A

```
a = 1
print(a)
```

INFO1_02_B

```
a = 0
print(a)
a = 100
print(a)
```

INFO1_02_C

```
a = 0
b = 100
print(a, b)
print(b, a)
```

INFO1_02_D

```
a = 0
b = 100
print(a, b)
a, b = b, a
print(a, b)
```

INFO1_03_A

```
a = input()
print(a)
```

INFO1_03_B

```
a = input()
b = input()
print(a)
print(b)
```

INFO1_03_C

```
a = input()
b = input()
c = input()
print(c)
print(b)
print(a)
```

INFO1_03_D

```
a, b, c, d = input().split()
print(d, c, b, a)
```

INFO1_03_E

```
a, b, c = input().split()
d, e = input().split()
print(e, d, c)
print(b, a)
```

INFO1_04_A

```
a = int(input())
b = int(input())
print(a + b)
```

INFO1_04_B

```
a = int(input())
b = int(input())
print(a - b)
```

INFO1_04_C

```
a = int(input())
b = int(input())
print(a * b)
```

INFO1_04_D

```
a = int(input())
b = int(input())
print(a // b)
```

INFO1_04_E

```
a = int(input())
b = int(input())
print(a % b)
```

INFO1_04_F

```
a = int(input())
print(a ** 10)
```

INFO1_04_G

```
a = int(input())
b = int(input())
c = int(input())
print(a - b * c)
```

INFO1_04_H

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
e = int(input())
print(int(a * b ** 3 + (c * d) / e - 100))
```

INFO1_04_I

```
a = int(input())
b = int(input())
print(a - a // b * b)
```

INFO1_05_A

```
a = int(input())
b = int(input())
print(a == b)
```

INFO1_05_B

```
a = int(input())
b = int(input())
print(a != b)
```

INFO1_05_C

```
a = int(input())
b = int(input())
print(a < b)
```

INFO1_05_D

```
a = int(input())
b = int(input())
print(a <= b)
```

INFO1_05_E

```
a = int(input())
b = int(input())
print(a + b >= 50)
```

INFO1_06_A

```
a = int(input())
print(not(a))
```

INFO1_06_B

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
print(a == b and c == d)
```

INFO1_06_C

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
print(a == b or c == d)
```

INFO1_06_D

```
a = int(input())
print(a == 2 or a == 3 or a == 5 or a == 7)
```

INFO1_06_E

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
print(a == b and c == d or a == c and b == d)
```

INFO1_06_F

```
a = int(input())
b = int(input())
c = int(input())
print(not(a and b) or c)
```

INFO1_07_A

```
x = int(input())
if x > 0:
    print("yes")
```

INFO1_07_B

```
x = int(input())
if x > 0:
    print("yes")
else:
    print("no")
```

INFO1_07_C

```
x = int(input())
if x > 0: print("1")
elif x < 0: print("-1")
else: print("0")
```

INFO1_07_D

```
x = int(input())
if x >= 80: print("A")
elif x >= 65: print("B")
elif x >= 50: print("C")
elif x >= 35: print("D")
else: print("F")
```

INFO1_07_E

```
a = int(input())
b = int(input())
if a > b: print(b)
else: print(a)
```

INFO1_07_F

```
a = int(input())
b = int(input())
c = int(input())
m = a
if m > b: m = b
if m > c: m = c
print(m)
```

INFO1_08_A

```
n = int(input())
i = 0
while i < n:
    print("hello")
    i += 1
```

INFO1_08_B

```
n = int(input())
for i in range(n):
    print(i + 1)
```

INFO1_08_C

```
while True:
    n = int(input())
    if n == 0:
        break
    print(n)
```

INFO1_08_D

```
while True:
    a = int(input())
    if a == 0: break
    if a > 0: print(a)
```

INFO1_08_E

```
n = int(input())
for i in range(n):
    for j in range(n):
        print("#", end = "")
    print()
```

INFO1_08_F

```
n = int(input())
for i in range(n):
    a = int(input())
    for j in range(a):
        print("#", end = "")
    print(" " + str(a))
```

INFO1_08_G

```
n = input() # useless
t = int(input())
sum = 0
while 1:
    sum += int(input())
    if sum >= t:
        break
print(sum)
```

INFO1_08_H

```
n = int(input())
cnt = 0
for i in range(n):
    a = int(input())
    if a == 0:
        print(cnt)
        cnt = 0
    else: cnt += 1
```

INFO1_09_A

```
N = int(input())
A = []
for i in range(N):
    A.append(input())
k = int(input())
print(A[k])
```

INFO1_09_B

```
N = int(input())
A = []
for i in range(N):
    A.append(input())
l = int(input())
r = int(input())
for i in range(l, r+1):
    print(A[i])
```

INFO1_09_C

```
N = int(input())
Q = int(input())
A = [0 for i in range(N)]
for i in range(Q):
    k = int(input())
    A[k] += 1
for i in range(N):
    print(A[i], end = " ")
print()
```

INFO1_09_D

```
N = int(input())
Q = int(input())
A = [0 for i in range(N)]
for i in range(Q):
    l = int(input())
    r = int(input())
    for k in range(l, r+1):
        A[k] += 1
for i in range(N):
    print(A[i])
print()
```

INFO1_09_E

```
N = int(input())
A = []
B = []
for i in range(2 * N):
    x = int(input())
    if x < 0: A.append(x)
    else: B.append(x)

for i in range(N):
    print(A[i], B[i])
```

INFO1_09_F

```
N = int(input())
check = [False] * 100001
for i in range(N):
    x = int(input())
    check[x] = True

for i in range(100001):
    if check[i]: print(i)
```

INFO1_10_A

```
c = input()
print(c)
```

INFO1_10_B

```
c = input()
print(ord(c))
```

INFO1_10_C

```
c = int(input())
print(chr(c))
```

INFO1_10_D

```
c = input()
if c.isdecimal():
    print("digit")
elif c.islower():
    print("lower")
elif c.isupper():
    print("upper")
else:
    print("other")
```

INFO1_10_E

```
s = input()
print(s)
```

INFO1_10_F

```
s = input()
print(len(s))
```

INFO1_10_G

```
s = input()
for c in s:
    print(c)
```

INFO1_10_H

```
s = input()
if s.isdecimal():
    s = int(s) + 1
print(s)
```

INFO1_11_A

```
col = input().split()
for s in col:
    print(s)
```

INFO1_11_B

```
col = input().split()
col.reverse()
for s in col:
    print(s)
```

INFO1_11_C

```
n = int(input())
for i in range(n):
    strs = input().split()
    for j in range(len(strs)):
        if j > 0: print(' ', end='')
        print(strs[j], end='')
    print()
```

INFO1_11_D

```
n = int(input())
for i in range(n):
    l = input().split();
    l.reverse();
    for j in range(len(l)):
        if  j > 0: print(' ', end='')
        print(l[j], end='')
    print()
```

INFO1_11_E

```
n = map(int, input().split())
for x in n:
    print(x + 1)
```

INFO1_11_F

```
n = int(input())
for i in range(n):
    a = list(map(int, input().split()))
    for j in range(len(a)):
        if j: print(" ", end = "")
        print(a[j] + i + 1, end = "")
    print()
```

INFO1_12_A

```
n, m = map(int, input().split())

A = []
for i in range(n):
    l = list(map(int, input().split()))
    A.append(l)
y, x = map(int, i
```

INFO1_12_B

```
n, m = map(int, input().split())

A = []
for i in range(n):
    l = list(map(int, input().split()))
    A.append(l)
y1, x1 = map(int, input().split())
y2, x2 = map(int, input().split())
for i in range(y2-y1+1):
    for j in range(x2 - x1 + 1):
        if j > 0: print(' ', end='')
        print(A[i + y1][j + x1], end='')
    print()
```

INFO1_12_C

```
n, m, q = map(int, input().split())

A = [[0 for i in range(m)] for j in
range(n)]

for i in range(q):
    y, x = map(int, input().split())
    A[y][x] = (A[y][x]+1)%2

for i in range(n):
    for j in range(m):
        if j > 0: print(' ', end='')
        print(A[i][j], end='')
    print()
```

INFO1_12_D

```
n, m, q = map(int, input().split())

A = [[0 for i in range(m)] for j in
range(n)]

for p in range(q):
    y1, x1, y2, x2 = map(int,
input().split())
    for i in range(y2-y1+1):
        for j in range(x2 - x1 + 1):
            A[i + y1][j + x1] += 1

for i in range(n):
    for j in range(m):
        if j > 0: print(' ', end='')
        print(A[i][j], end='')
    print()
```

INFO1_12_E

```
X, Y, Z = map(int, input().split())
N = int(input())

A = [[[0 for i in range(Z+1)] for j in
range(Y+1)] for k in range(X+1)]

for i in range(N):
    x, y, z = map(int, input().split())
    A[x][y][z] = 1

Q = int(input())
for i in range(Q):
    x, y, z = map(int, input().split())
    print(A[x][y][z])
```

INFO1_13_A

```
def add(x):
    return x + 1

x = int(input())
print(add(x))
```

INFO1_13_B

```
def printFormat(n, c):
    for i in range(n):
        print(c, end = "")
    print()

n = int(input())
c = input()
printFormat(n, c)
```

INFO1_13_C

```
import random

def getElement(a):
    return random.choice(a)

n = int(input())
a = list(map(int, input().split()))
print(getElement(a))
```

INFO1_13_D

```
def printFormat(a):
    print("(", end = "")
    for i in range(len(a)):
        if i: print(", ", end = "")
        print(a[i], end = "")
    print(")")

n = int(input())
a = list(map(int, input().split()))
printFormat(a)
```

INFO1_13_E

```
def getList(n):
    return range(n)

def printFormat(a):
    print("(", end = "")
    for i in range(len(a)):
        if i: print(", ", end = "")
        print(a[i], end = "")
    print(")")

n = int(input())
a = getList(n)
printFormat(a)
```

INFO1_13_F

```
def prodList(a, x):
    b = []
    for m in a:
        b.append(m * x)
    return b

def printFormat(a):
    print("(", end = "")
    for i in range(len(a)):
        if i: print(", ", end = "")
        print(a[i], end = "")
    print(")")

n = int(input())
a = list(map(int, input().split()))
x = int(input())
b = prodList(a, x)
printFormat(a)
printFormat(b)
```

INFO1_13_G

```
def addList(a, b):
    c = []
    for i in range(len(a)):
        c.append(a[i] + b[i])
    return c

def printFormat(a):
    print("(", end = "")
    for i in range(len(a)):
        if i: print(", ", end = "")
        print(a[i], end = "")
    print(")")

n = int(input())
a = list(map(int, input().split()))
b = list(map(int, input().split()))
c = addList(a, b)
printFormat(a)
printFormat(b)
printFormat(c)
```

INFO1_14_A

```
def search(A, key):
    for v in A:
         if v == key:
             return True
    return False

n = int(input())
A = list(map(int, input().split()))
key = int(input())

if search(A, key):
    print("Yes")
else:
    print("No")
```

INFO1_14_B

```
def search(A, key):
    for i in range(len(A)):
         if A[i] == key:
             return i
    return -1

n = int(input())
A = list(map(int, input().split()))
key = int(input())

print( search(A, key))
```

INFO1_14_C

```
def search(A, key):
    for i in range(len(A)):
         if A[i] == key:
             return True
    return False

n = int(input())
A = list(map(int, input().split()))
q = int(input())
for i in range(q):
    key = int(input())
    if search(A, key):
        print("Yes")
    else:
        print("No")
```

INFO1_14_D

```
def bsearch(A, key):
    l = 0
    r = len(A)
    while(l < r):
        print(f"[{l}, {r})")
        m = (l + r) // 2
        if A[m] == key: break;
        if A[m] > key:
            r = m
        else:
            l = m + 1

n = int(input())
A = list(map(int, input().split()))
key = int(input())
bsearch(A, key)
```

INFO1_14_E

```
def bsearch(A, key):
    l = 0
    r = len(A)
    while(l < r):
        m = (l + r) // 2
        if A[m] == key: return True
        if A[m] > key:
            r = m
        else:
            l = m + 1
    return False

n = int(input())
A = list(map(int, input().split()))
q = int(input())
for i in range(q):
    key = int(input())
    if bsearch(A, key):
        print("Yes")
    else:
        print("No")
```

INFO1_14_F

```
def search(A, key):
    for v in A:
        print(f" {v}", end="")
        if v == key: break
    print()

def bsearch(A, key):
    l = 0
    r = len(A)
    while(l < r):
        m = (l + r) // 2
        print(f" {A[m]}", end="")
        if A[m] == key: break;
        if A[m] > key:
            r = m
        else:
            l = m + 1
    print()
n = int(input())
A = list(map(int, input().split()))
key = int(input())
search(A, key)
bsearch(A, key)
```

INFO1_14_G

```
n = int(input())
print(n)
cnt = 0
while n > 0:
    cnt += 1
    n //= 2

print(cnt)
```

INFO1_15_A

```
def check(A):
    pre = -1
    for v in A:
        if pre > v:
            return False
        pre = v
    return True

n = int(input())
A = list(map(int, input().split()))

if check(A):
    print("Yes")
else:
    print("No")
```

INFO1_15_B

```
n = int(input())
A = list(map(int, input().split()))
a, b = map(int, input().split())

A[a], A[b] = A[b], A[a]

for i in range(len(A)):
    if i > 0: print(' ', end="")
    print(f"{A[i]}", end="")
print()
```

INFO1_15_C

```
n = int(input())
A = list(map(int, input().split()))
a, b = map(int, input().split())

T = A[a:(b+1)]
T.reverse()
A[a:(b+1)] = T

for i in range(len(A)):
    if i > 0 : print(' ', end="")
    print(f"{A[i]}", end="")
print()
```

INFO1_15_D

```
n = int(input())
A = list(map(int, input().split()))
k = int(input())

T1 = A[0:k]
T2 = A[k:]
for v in T1:
    T2.append(v)

for i in range(len(T2)):
    if i > 0: print(" ", end="")
    print(f"{T2[i]}", end="")
print()
```

INFO1_15_E

```
n = int(input())
A = list(map(int, input().split()))
A.sort()
for v in A:
    print(f" {v}", end="")
print()
```

INFO1_15_F

```
n = int(input())
A = list(map(int, input().split()))
A.sort()
A.reverse()
for v in A:
    print(f" {v}", end="")
print()
```

INFO1_15_G

```
def check(A):
    N = len(A)
    if N == 1: return 0
    ok = True
    for i in range(N-1):
        if A[i] != A[i+1]: ok = False
    if ok: return 0
    ok = True
    for i in range(N-1):
        if A[i] >= A[i+1]: ok = False
    if ok: return 2
    ok = True
    for i in range(N-1):
        if A[i] > A[i+1]: ok = False
    if ok: return 1
    ok = True
    for i in range(N-1):
        if A[i] <= A[i+1]: ok = False
    if ok: return -2
    ok = True
    for i in range(N-1):
        if A[i] < A[i+1]: ok = False
    if ok: return -1

n = int(input())
A = list(map(int, input().split()))

print(check(A))
```