Python で学ぶ プログラミング

東洋大学経営学部「プログラミング実習講義」テキスト

関　勝寿

はじめに

このテキストは、東洋大学経営学部専門科目「プログラミング実習講義」の教材です。この授業は、プログラミング未経験の学生がプログラミングの経験を通してプログラミングに必要な論理的思考力を身につけ、アルゴリズムとデータ構造の理解を深めることを目的としています。

私が東洋大学に着任したのは 2008 年度で、それまで東洋大学経営学部の野中誠先生が担当されていた「数理・情報実習講義 B」という科目を引き継ぎました。内容は Java プログラミングであり、野中先生作成のテキストを引き継いで授業をしました。その後、2014 年度からは「情報処理実習 D」2019 年度からは「プログラミング実習講義」と科目名が変わり、当初のテキストに改訂を重ねながら授業を続けてきました。2018 年度までは履修者数が 10 名程度でしたが、2019 年度には履修希望者が 299 名に急増しました。科目名に「プログラミング」が入っただけでこれほど履修者が増えるほどにプログラミングの関心が高まっているのかと驚きました。

そして 2020 年度からは、Python を教える授業に変えることとしました。その理由は、統計や AI に使われる言語として Python の人気が高まり、学生からも「授業で Python を教えてほしい」という声が寄せられるようになったこと、私自身も、2015 年に Python の勉強を始め、日常的に Python を使うようになり、Python を気に入っていることなどです。テキスト中のプログラムを Java から Python に書き直したことで、全体的にコードもすっきりと短くなり、初学者に理解しやすくなったのではないかと思います。また、学生が自宅の PC でも学習しやすいように、オンラインのプログラム実行環境 Paiza を使うこととしました。

履修者の多くはプログラミングの初学者ですが、プログラミングの経験がある学生もいます。そのような学生にとっては初学者向けの内容ばかりだと物足りないので、ところどころ「発展」としてさらに深く学びたい学生が学習をするためのきっかけを与えています。なお、履修希望者が増えていることから、2023 年度からは「プログラミング実習講義」のコース数を増やしています。このテキストは関が担当教員となっているコースのテキストです。

テキストの多くの部分は私が書いたものですが、野中先生作成の図面や解説も含まれています。野中先生の許可を得て 2020 年度版のテキストからインターネットに公開します。

本書のダウンロード： https://sekika.github.io/toyo/python/

2020 年 9 月 5 日 公開
2024 年 4 月 2 日 更新

関　勝寿

目 次

第 1 回　ガイダンス 4

第 2 回　プログラミング言語について 6

第 3 回　PYTHON の基礎 10

第 4 回　計算プログラム 16

第 5 回　条件分岐 19

第 6 回　繰り返し処理 23

第 7 回　コンテナのデータ構造 27

第 8 回　再帰的アルゴリズム 31

第 9 回　整列アルゴリズム（バブルソート） 35

第 10 回　整列アルゴリズム（クイックソート） 39

第 11 回　探索アルゴリズム 43

第 12 回　オブジェクト指向プログラミング 47

第 13 回　スタックとキューのデータ構造 51

第 14 回　数値計算アルゴリズム（モンテカルロ法） 55

第 15 回　数値計算アルゴリズム（ニュートン法） 57

第１回　ガイダンス

「プログラミング実習講義」の授業の進め方について説明します。

【講義の目的・内容】
スマホやパソコンをはじめとして、エアコン、テレビ、照明、電子レンジ等の家電製品や自動車などの製品には、IoT や AI などの高度な情報技術が使われるようになり、情報技術を活用するスキルである「IT 力」の重要性が高まっている。IT 力を高めるためには、情報技術の基盤である「プログラミング」の理解が不可欠である。なぜならば、あらゆる IT 機器はプログラムにしたがって動いているためである。2020 年度からすべての小学校においてプログラミング教育が必修化され、プログラミング的思考は義務教育レベルの教養となった。
このように、プログラミングの専門家にならなくても、プログラムがどのように動いているかという仕組みを知ることは、現代社会を生きる上で必須の基礎的な素養である。
経営学科の「経営情報・分析メソッド」科目群に位置づけられるこの実習講義では、プログラミング未経験者がプログラミングの経験を通してプログラミングに必要な論理的思考力を身につけ、アルゴリズムとデータ構造の理解を深めることができるように、統計、AI の分野で近年人気が上昇しているプログラミング言語 Python のプログラムを動かしながら学ぶ。

【講義スケジュール】
日程は ToyoNet-ACE を確認すること。

第１回 ガイダンス
第２回 プログラミング言語について
第３回 Python の基礎
第４回 計算プログラム
第５回 条件分岐
第６回 繰り返し処理
第７回 コンテナのデータ構造
第８回 再帰的アルゴリズム
第９回 整列アルゴリズム（バブルソート）
第 10 回 整列アルゴリズム（クイックソート）
第 11 回 探索アルゴリズム
第 12 回 オブジェクト指向プログラミング
第 13 回 スタックとキューのデータ構造
第 14 回 数値計算アルゴリズム（モンテカルロ法）
第 15 回 数値計算アルゴリズム（ニュートン法）

【履修登録】
履修登録は、必ず履修登録期間中にすること。抽選や追加履修登録などの情報にも注意すること。なお「プログラミング実習講義」は複数の教員により複数のコースが開講されている。ToyoNet-G 参照。

【テキスト】
テキストはこの PDF ファイル（「Python で学ぶプログラミング」）であり、
https://sekika.github.io/toyo/python/ からダウンロードする。本書全体を一括してダウンロードすることも、章ごとに表示することも可能である。まとめて**印刷**しておくことを推奨する。履修者に印刷体を配布する可能性もあり。ACE での案内を参照すること。

【指導方法】

1. 授業はテキストに沿って PC 教室で講義スケジュールの通りに進める。
2. 学生は、PC でプログラムを実行する。その際には、ウェブブラウザ上でプログラムを実行できる **Paiza** というサイトを使う。ウェブブラウザを使うことができれば、PC（Windows, Mac, Linux）でもタブレットでも受講可能である。
3. あらかじめ用意した半完成プログラムの空欄に、適切なプログラムコードを記入して完成させる形式での課題を中心とする。
4. **課題**は ToyoNet-ACE のレポートまたは小テストから提出する。授業時間中に提出が完了した者は帰って良い。授業時間中に提出が完了しなかった者は、課題の提出期限は**授業 3 日後の 23:00** を原則とする。理由があって授業に出席できなかった場合には、テキストを読んで課題を提出すれば出席したものとする。
5. 提出期限をすぎた場合は理由の如何を問わずに提出を受け付けない。ネットの不調などで提出できなかった場合も同様とするため、提出期限ぎりぎりで提出しようとしないこと。
6. 提出されたプログラムは教員の PC で実行し、実行結果に応じて採点され、提出期限終了後数日以内に ToyoNet-ACE の「**成績**」から採点結果を閲覧できるようになる。詳しい採点基準については第 3 回の授業で示す。

【Python 実行環境の確認】

1. https://paiza.io/projects/ziw0mlroy8_4J3ZzBRDX8A にアクセスする。
2. 「実行」ボタンを押す。　「出力」に「グー」「チョキ」「パー」のいずれかが出ることを確認する。「実行」ボタンを押すたびに変わる。
3. この画面を直接編集してプログラムを書き換えることが可能である。たとえば「グー」を「大吉」に「チョキ」を「中吉」に「パー」を「小吉」に書き換えて実行してみよう。

【成績評価】

課題を提出することをもって出席とし、課題の平均点によって評価する。課題が未提出の場合にはその回の授業を欠席したものとし、課題の点数が 0 点となる。課題の提出が 3 分の 2 以下の場合、すなわち未提出回数が 5 回以上の場合は、単位を認めない。期末試験は実施しない。このテキストの「第 1 回 ガイダンス」を熟読すること。**ガイダンスの内容を理解していないことによる成績判定に関する申し立ては受け付けない。**

４年生を特別扱いすることは一切ないので「卒業できないから単位下さい」という者は、はじめからこの授業を履修しないこと。

【担当教員】

・名前：関　勝寿（せき　かつとし）
・ ホームページ：https://sekika.github.io/toyo/
自作プログラム（土壌物理学、数学、パズル）あり
・ ToyoNet-G の教員プロフィールに、メールアドレスが書かれている。メールを送るときは、東洋大学の toyo.jp のメールアドレスを使って下さい。また、メールには科目名を入れて、用件がすぐに分かる適切な件名（例：「プログラミング実習講義」の課題について）をつけて下さい。また、氏名と学籍番号を必ず、できれば学部、学科、学年を記して下さい。

第 2 回　プログラミング言語について

プログラミング言語の役割について学ぶ。

【ハードウェアとソフトウェア】
パソコン、スマホ、テレビ、冷蔵庫、洗濯機、インターフォン、ウォシュレット、その他あらゆる電子機器には、ハードウェアとソフトウェアがある。
ハードウェア（hardware）とは、物理的な構成要素である。たとえば、モニタ、キーボード、マウス、CPU（中央処理装置）、メインメモリ（主記憶装置）、外部記憶装置（ハードディスク、SSD、USB メモリ）、ケース、メインボード（マザーボード）、などである。
ソフトウェア（software）とは、ハードウェアとしてのコンピュータを動作させるための、プログラム、データなどの総称[1]であり、ハードウェアのような物理的実体はない。コンピュータはソフトウェアの指示に従って、その目的とする処理を行う。ソフトウェアを取り替えることで、コンピュータはさまざまな処理を行うことができる。「コンピュータ、ソフトがなければ ただの箱」と言われるように、コンピュータにとってソフトは必要不可欠なものである。

【プログラム】
プログラミング言語とは、コンピュータに理解できるよう、動作手順を表現してコンピュータに伝える人工的な言語であり、動作手順をプログラミング言語の言葉として表現したものがプログラムである[2]。

【基本ソフトウェア】
基本ソフトウェア（OS: Operating System）とは、コンピュータを利用するための基本的な制御やプロセス管理などを行うソフトウェアであり、基本ソフトウェアの上で応用ソフトウェアが動く。基本ソフトウェアは、ハードウェアを制御する基本的なソフトウェアを用意して、応用ソフトウェアとハードウェアとの仲立ちをする[3]。

【パソコンの主な OS】
- MS-DOS（Microsoft Disk Operating System）
  - Microsoft 社が 1981 年に開発した IBM PC 用の OS。
  - その後、バージョンアップを重ね、IBM PC/AT 互換機パソコン用 OS の業界標準となった。
- macOS
  - Apple 社のパソコン Macintosh 用の OS。OS の名称は Mac OS, Mac OS X, OS X などと変更され、2016 年に開発されたバージョン 10.12 から macOS となった。
  - 早くから優れた GUI を備え、Windows などに大きな影響を与えた。
- Windows
  - Microsoft 社が MS-DOS の後継として開発した OS。
  - 1986 年に最初のバージョン 1.0 が発表された後、3.1, 95, 98,XP,Vista,7,8,10,

---

[1] JIS の定義では、ソフトウェアは「データ処理システムを機能させるための、プログラム、手順、規則、関連文書などを含む知的な創作」とされていて、この場合、規則や関連文書も「ソフトウェア」に含まれる。

[2] ソフトウェアは、コンピュータに関わるすべての知的情報で、プログラムは、コンピュータの動作手順を表したものに限定されるため、ソフトウェアの方がプログラムよりも広い概念で、プログラムは、ソフトウェアに含まれる。

[3] JIS の定義では、「プログラムの実行を制御するソフトウェアであって、資源割振り、スケジューリング、入出力制御、データ管理などのサービスを提供するもの」とされている。

11 などバージョンアップを重ねている。

- UNIX
  - AT&T のベル研究所で 1969 年に開発されたワークステーション用 OS。
  - 大学や研究機関を中心に広く普及。
- Linux （リナックス）
  - フィンランドの大学生リーナス・トーバルズが、UNIX を真似て原型を開発。
  - オープンソースソフトウェアとして公開され、改変が自由に行われている。

【スマホの主な OS】

- Android
  - Google が開発している OS で、スマホのメーカーがカスタマイズしている。
- iOS
  - Apple が iPhone や iPad などの自社製品のために開発している OS。

【応用ソフトウェア】

応用ソフトウェア（アプリケーションソフトウェア application software; アプリ app）とは、一般のユーザーが利用する目的で作られたプログラムである。プログラムの開発元が作成したアプリをユーザーがパソコンあるいはスマホの OS にインストールし（あるいは、製品購入時にすでにインストールされている）、実行する。たとえば次のようなものがある。

1. ワープロソフトウェア
2. 表計算ソフトウェア
3. データベースソフトウェア
4. プレゼンテーションソフトウェア
5. ウェブブラウザ
6. グラフィックスソフトウェア
7. ユーティリティソフトウェア
8. 業務ソフトウェア
9. ゲームソフトウェア

誰かが作ったプログラムをインストールして実行するだけでなく、自ら応用ソフトウェアのプログラミングをして実行することもできる。この授業では自らが作成したプログラムを実行するという経験をする。

【プログラミング言語の種類】

1. 機械語（マシン語）：CPU が解釈し実行できる唯一の言語。機械語のプログラムは、CPU の１単位の動作を表現する命令を並べたもの。
2. アセンブリ言語：機械語の命令を人間に分かりやすいような表現で記述したもの。アセンブラにより、機械語に変換してから実行される。
3. 高水準言語：自然言語に近く、人間の考えを表現しやすいプログラミング言語。英語に近い記述ができる。

高水準言語によって書かれたプログラムの実行方法には、コンパイラによって機械語のプログラム（実行ファイル）に変換してから実行する方法と、インタプリタを起動して、インタプリタにプログラム（ソースファイル）を読み込ませて実行する方法がある[1]。

[1] 最初はインタプリタとして動作して、コンパイルが完了すると高速に実行する動的コンパイルなどの手法もある。

【主な高水準言語】
特に有名なものを選んで簡単に紹介する。

1. FORTRAN：1954 年に IBM が開発。コンピュータにおいて広く使われた最古の高水準言語であり、特に科学技術計算の分野では今でも利用されている言語。
2. LISP：1958 年に計算機科学者のジョン・マッカーシーが考案。現在広く使われているプログラミング言語の中では、FORTRAN に次いで古い。S 式というリスト構造の表記法を使うことが特徴であり、そのために大量の括弧を使うところが印象的である。
3. COBOL：1960 年に CODASYL が開発。汎用コンピュータでの事務処理用として現在も広く利用されている言語。
4. BASIC：1970 年代以降にパソコン用の言語として広く使われた言語。Visual Basic などへ発展し、Microsoft Office には VBA (Visual Basic for Applications)として搭載されている。
5. Pascal: 1970 年に構造化プログラミングとして設計され、特に 1980 年代に人気を博した言語。Turbo Pascal や Delphi などの統合開発環境が発売された。
6. C：1972 年に開発されて UNIX の記述に用いられるようになり、現在も幅広く利用されている言語。ポインタによってメモリを直接指定してアクセスするといったような、高水準言語の中では低水準な処理ができた。その後、様々な C 系言語が派生した。
7. Smalltalk：オブジェクト指向言語として最初に普及した言語。1972 年に開発が開始され、1980 年に公開された。
8. C++：C 言語をベースにして、オブジェクト指向などの様々な機能を取り込んだ言語。C 言語の上位互換とされるが、厳密には異なる。1983 年に公開された。
9. Objective-C: 1983 年に Brad Cox らが開発。その後スティーブ・ジョブズの NeXT コンピュータの主力言語となり、権利を買い取る。さらにアップルが NeXT 社を買収し、Mac OS X の Cocoa フレームワークのコア言語として採用した。
10. Perl: ラリー・ウォールが 1987 年に開発したスクリプト言語。ウェブアプリケーション、システム管理、テキスト処理などのプログラムを書くために現在も広く用いられている。非常に高いレベルで後方互換性が維持されることで定評がある。
11. Haskell: 純粋関数型プログラミング言語。習得の難易度は高いがバグが発生しにくいとされている。1987 年にアメリカのポートランドで開かれた関数型プログラミング言語に関する学術会議で委員会が結成され、1990 年に作成された。
12. Python：1991 年に公開されたスクリプト言語。読みやすく簡潔にコードが書けるように設計されている。この授業で採用するプログラミング言語であるため、この後にさらに詳しく紹介する。
13. Ruby: 1995 年にまつもとゆきひろが開発したスクリプト言語。日本で開発されたプログラミング言語としてははじめて国際電気標準会議で国際規格に認証された。ウェブアプリケーションフレームワークの Ruby on Rails が広く使われている。
14. Java：1995 年に Sun Microsystems が開発し、Oracle が開発を引き継いでいる。Smalltalk と C++をベースとした、機種非依存のオブジェクト指向言語。企業の業務システム、PC やスマホのアプリ、家電製品の組み込み、など幅広い用途に利用される。環境を選ばない汎用性の高さから需要が高く、システム開発では最も使われている言語の一つである。
15. JavaScript: 1995 年に Netscape Communications が開発し、Netscape Navigator というウェブブラウザで実装されたスクリプト言語。主要なウェブブラウザ上で動作するため、動的なウェブサイト構築やウェブサイト上でのアプリケーション開発に用いられる。Java と混同されがちであるがまったく異なる。
16. PHP: ラスマス・ラードフが 1995 年に公開したスクリプト言語。ウェブアプリケーション開発で使われている言語。レンタルサーバーで動作するところが多く、初心者が習得しやすいと思われていることから、一定の人気を保っている。

17. C#: Microsoft が 2000 年に開発し、C、C++、Java、Delphi 他多くの言語の影響を受けている。Microsoft のフレームワーク .NET Framework で動作する。
18. Scratch: 子供に楽しくプログラミングを学習させることを目的に 2006 年に MIT メディアラボが開発した。パレットにブロックを並べていくというインターフェイスを採用しているため、敷居が低くなっている。プログラミングの必修化にともない、日本でも教育業界で普及し、学校の授業でも取り入れられている。
19. Rust: C 言語、C++に代わるシステムプログラミング言語として、性能、メモリ安全性、安全な並行性を目指して設計された。2006 年に Mozilla 従業員の Graydon Hoare が開発を始め、2009 年から Mozilla Research の公式プロジェクトとなった。
20. Go: 2009 年に Google が発表した。golang とも呼ばれる。並行処理機能が優れていることなどからクラウドアプリ開発での人気が高い。
21. 2010 年代前半に開発された Dart, TypeScript, Swift, Kotlin, Julia は、いずれもこれまでの多くのプログラミング言語の良い特徴を取り入れて使いやすくされていることから人気が高く、今後の発展が期待されている。いずれもオープンソースである。
22. クロスプラットフォームアプリ開発フレームワークとして、Dart ベースの Flutter, JavaScript, TypeScript ベースの React Native, Cordova, Ionic, C#ベースの .NET MAUI (旧 Xamarin), Unity などがよく使われている。

【Python について】

Python (パイソン) は、グイド・ヴァン・ロッサム[1]が 1991 年に公開したプログラミング言語であり、世界中のプログラマーのコミュニティによって開発が続けられ、2001 年には非営利団体 Python ソフトウェア財団 (PSF) が立ち上がり、フリーソフトウェアの PSF ライセンスで配布されている。

Python は多くの環境で動作し、簡潔にコードを書けることが特徴として挙げられる。科学計算でよく使われ、数値計算ライブラリの NumPy や科学計算ライブラリの SciPy、データ解析ライブラリの pandas、グラフ描画ライブラリの Matplotlib などを使うことで、高度なデータ処理や統計解析をすることができる。

特に、近年の AI ブームは Python の人気上昇を加速させている。Google では、社内の公式言語として C/C++、Java、JavaScript、Python、Go が使われている。Google が開発しオープンソースで公開している TensorFlow という機械学習のためのソフトウェアライブラリが、AI のプログラミングに便利である。

このように学術的な分野で人気が高い Python であるが、実務でも Django のような Web アプリケーションフレームワークによって Web サイトの構築ができることから、よく使われている。YouTube、Instagram、Dropbox は Python で開発されたとされている。

この授業では、Scratch のような教育用のプログラミング言語を使わずに、Python を学習する。この授業をきっかけとしてプログラミングを本格的に学び、これからの大学での学習や就職後に使うスキルとして役立てるためには、よく使われている言語で習熟する方が良いと考えるためである[2]。まずは Python で簡単なプログラミングができるようになることで、必要に応じて他の言語も自習により習得できるようになるであろう。

【★課題】

ToyoNet-ACE の小テストを提出期限までに受験すること。提出期限をすぎた場合は理由の如何を問わずに提出を受け付けない。

---

[1] Guido van Rossum – Personal Home Page https://gvanrossum.github.io/

[2] プログラミング言語の人気については https://redmonk.com/ / https://www.tiobe.com/tiobe-index/ / https://spectrum.ieee.org/ / https://youtu.be/Og847HVwRSI などを参照

第 3 回　Python の基礎

Python のプログラムを実行する方法と、エラーの修正方法について学ぶ。

【Python のバージョンについて】
Python のバージョン番号は A.B.C や A.B のように付けられている。A はメジャーバージョン番号で、言語の仕様に関わる重要な変更のときに上げられる。B はマイナーバージョン番号で、それほど大きくはない変更のときに上げられる。C はマイクロレベルで、バグを修正するときに上げられる。
Python は 1991 年にバージョン 0.9 のソースコードが公表されたときにはすでにオブジェクト指向言語としての特徴を備え、1994 年にバージョン 1.0 が公開されたときには、関数型言語の基本であるラムダ計算などが組み込まれた。2000 年にバージョン 2.0 が公開され、リスト内包表記やメモリを効率的に管理するガーベージコレクションなどの機能が実装され、一躍メジャーな言語になった。2008 年には Python 3.0 がリリースされた。現在広く使われている Python のメジャーバージョンは 2 と 3 であり、それぞれ Python 2、Python 3 と書かれる。2020 年 1 月 1 日に Python 2 の開発は停止したため、この授業では Python 3 を使って学習する。

【Python のセットアップ】
Python でプログラミングをするためには、Python をインストールする必要がある。その方法は、Windows, macOS, Linux などの UNIX プラットフォームなどの OS によって異なり、詳しくは Python ドキュメントページで解説されている。

https://docs.python.org/ja/3/

基本的には、Python のページ https://www.python.org/ から環境に合わせたインストーラーをダウンロードしてインストールする。プログラムは文字コード UTF-8 のテキストファイルで作成する。
この授業では、Python のインストールはせずにオンライン実行環境を使って実習をする。

【Python の実行方法】
Python の実行方法には、何通りかの方法がある。1 つ目は、ターミナルから起動する方法である。Windows ではコマンドプロンプトまたは PowerShell を起動して py と入力する。Mac ではターミナルを起動して python と入力する。この方法では、Python が対話モード（インタラクティブモード）で起動する。最初に Python のバージョンなどが表示されて、>>> のような「プロンプト」が表示されて、コマンドの入力待ちをする状態になる。直接 Python のコマンドを入力して実行することができる。Ctrl+D を押すか exit() と入力する事で、対話モードを終了することができる。細かい説明は省略するが、複数の Python のバージョン（たとえばバージョン 2 とバージョン 3 など）がインストールされている場合には、バージョンを指定して起動する場合もある。対話モードの Python の中で現在使っているバージョンを確認するには、import sys; print(sys.version) と入力する。
統合開発環境 IDLE を起動することもできる。Windows の場合はスタートメニューから IDLE を選ぶ。Mac の場合には、ターミナルから idle と入力する。
Jupyter Notebook によって、プログラム、メモ、グラフなどの実行結果をまとめて管理する手法は、特に機械学習やデータサイエンスの分野では主流となっている。

【オンライン実行環境 Paiza】
通常は PC にプログラム開発環境をインストールしてプログラムを開発するものであるが、ウェブブラウザ上でプログラムを開発できる。たとえば、次のようなサイトがある。

(1) paiza.IO （ https://paiza.io/ja ）
　日本語対応。
(2) Ideone （ https://ideone.com/ ）
　対応している言語が多い。
(3) Replit （ https://replit.com/ ）
　対応している言語が多く機能が豊富。Matplotlib のグラフ出力も見ることができる。
(4) Google Colaboratory （ https://colab.research.google.com/ ）
　Google のクラウド上で Jupyter Notebook の Python 実行環境を利用できる。

この授業では、OS などの環境の違いを吸収するために、Paiza を使ってプログラムを編集し、実行する。

【Hello プログラム】
それでは、早速 Paiza を使って Hello プログラムを実行してみよう。

★マークの箇所は、実際にやってみること。

★ https://paiza.io/ja にアクセスする。
★ 言語が日本語になっていなければ日本語を選ぶ。

以下はログインしなくても続けることができるが、ログインすることでそれまでに自分が作成したプログラムを一覧する機能があるので、今後の効率的な学習のためには、サインアップしてログインすることを推奨する。

★ 緑色の「コード作成を試してみる（無料）」ボタンを選ぶ。
★ 左上から言語を選択する。「Python3」を選ぶ。
★ 次のプログラムを打ち込む。日本語入力はオフにすること。

```
print("Hello!")
```

このように四角で囲まれた箇所はプログラムをあらわす。1 行目と 2 行目には # （ハッシュ）で始まる行があらかじめ入っているが、 # で始まる行は「コメント」でありプログラムの実行に影響を与えないので、そのまま残しておいても、消しても構わない。なお、最初の行に書かれている「# coding: utf-8」は、ファイルの文字コードが UTF-8 であることを示していて、Python 2 ではこの記述が必要であったが、Python 3 では UTF-8 が標準で使われるので、この記述を省略することが可能である。

★ 「実行（Ctrl-Enter）」ボタンを押す。「出力」に「Hello!」と表示されることを確認する。「Hello!」以外の文字が表示された時、あるいは「実行時エラー」に英語のメッセージが表示されたときには、プログラムを修正する。その方法については、【エラーのデバッグ】で詳しく解説する。
★ デバッグが終わって、実行したら「Hello!」と表示されるようになったら、【★課題】で指示されている通りに課題の提出をする。

【プログラムの解説】
print() は、Python に標準で用意されている「組み込み関数」の一つである。「関数」は、数学の関数に似ている。たとえば、y = 2x という関数は、x という引数に対して y と

いう値を返す関数であり、そのような関数が定義されていれば、x=2 という引数を与えることで、y=4 という結果が得られる。このように、なんらかの値を「引数」として与えてプログラムに処理をさせて、なんらかの結果（戻り値）を返すものを「関数」と言う。ただし、「戻り値」がない場合もあり、print は戻り値がない関数である。

```
print(引数)
```

のように指定すると、引数の内容を文字列として出力する（さらに詳しい使い方については、ここでは触れない）。ここで「出力する」というのは、結果を表示するということで、この場合は Paiza の「出力」に表示されるが、どこに出力するか（たとえばファイルに出力するか）といったことを制御することも可能である。

print(引数)の「引数」のところには、"Hello!"と書かれている。ここで、”は二重引用符であり、いわゆる「半角文字」で入力する必要がある。日本語入力をオンにしていると、「“」という全角文字が入力されてしまいエラーになる。

Python では、二重引用符 “ あるいは引用符 ‘ で囲まれた文字の並びは「文字列」というデータとして解釈される。ここで Hello! を二重引用符で囲んで “Hello!” と書くことで、Hello!という文字列として解釈され、それが引数として print 関数に渡されて、Hello! という文字列が出力される。Hello! を二重引用符で囲まずに print(Hello!) として実行すると、「!」の意味を解釈できないとして、次に説明する文法エラー（SyntaxError）となる。

【エラーのデバッグ】

「デバッグ」とは、プログラムのおかしいところ（バグ）を見つけて修正する作業である。バグとは「虫」を意味し、虫取りというような意味である。プログラミングは、何度も「思い通りに動かない」ことに遭遇し、ミスを修正するという作業を繰り返す。今回のように、たった 1 行のプログラムであっても、最初はなんらかのエラーがでてうまく動かないことがある。そのときに「どこがエラーであるかを見つける」ことが、プログラミングの重要な技術である。なお、【Python の実行方法】で紹介した統合開発環境 IDLE にはデバッグ機能があり、効率良くデバッグをすることができる[1]。

ここでは、典型的なエラーを実際に発生させてみて、どのようなミスをするとどのようなエラーが表示されるかを確認する。それがわかれば、エラーの表示を見て修正する作業が容易となる。

(1) ケース 1：最後の)を全角文字にしてしまった。

Python を含む多くのプログラミング言語では、ASCII の文字[2]が使われるため（文字列を指定したりコメントを書いたりする場合を除く）、非 ASCII 文字、つまり ASCII ではない文字は「無効な文字」であるとされる。日本語入力によって変換された文字はそのような無効な文字になる。この授業では ASCII の文字を「半角文字」、日本語入力された非 ASCII 文字を「全角文字」と呼ぶ[3]。たとえば、数字には半角文字の「1234567890」と全角文字の「１２３４５６７８９０」がある。アルファベットや()”,などの記号にも対応する全角文字があ

---

[1] Python の IDLE でデバッグを行う方法 https://gammasoft.jp/python/debugging-python-idle/

[2] ASCII（American Standard Code for Information Interchange; アスキー）は、コンピュータで最もよく使われている文字コード体系で、7 桁の 2 進数（10 進数では 0 から 127 まで）に文字コードが割り当てられている。キーボードで日本語入力モードを ON にしないで入力される文字（123…のような数字、abc…のようなアルファベット、()のような記号）は、ASCII の文字コードであらわされる。

[3] 厳密には不正確だが、話をわかりやすくする。

る。「日本語入力はオフにすること」という指示を実行していれば全角文字が入力されることはないはずであるが、ここでは試しに、最後の)を全角文字）にして実行をしてみると、「実行時エラー」に次のように表示される。

```
  File "Main.py", line 1
    print("Hello!")
                 ^
SyntaxError: invalid character in identifier
```

このメッセージには、2 つの情報が含まれている。それは「どこにエラーがあるか」そして「どのような種類のエラーがあるか」ということである。まずは、最初の行に

```
  File "Main.py", line 1
```

とあり、これは Main.py というファイルの 1 行目にエラーがあることを示している。今回はたった 1 行のプログラムであるためエラー箇所を見つけるのは簡単であるが、プログラムの行数が増えたときには、まずどこの行でエラーが発生したのかを確認することが大切である。さらに、次の行には

```
    print("Hello! “)
                  ^
```

と表示されている。^ ここで「 ^ 」マークの位置が、エラーが発生している場所を示している。つまり、print 文が終わったところでエラーが発生している、ということがわかる。この位置の前後をよく見ることで、エラーを発見できる。

今回の授業では、テキストに書かれているプログラムをそのまま打ち込めば、正常に実行ができる。そうなっていないということは、必ずどこかに打ち間違いがある。単語の綴りを間違えていたり、アルファベットの大文字と小文字を間違えていたりしている。したがって、もしエラーが発生したときには、エラーの原因がわからなくても、エラーが発生した箇所をよくみて、その前後を確認する、あるいは打ち直すことで、エラーを修正することができる。さらに、次に表示されている

```
SyntaxError: invalid character in identifier
```

という表示の意味がわかれば、さらにエラーの原因がつかみやすくなる。より複雑なプログラムをデバッグするときには、エラーの箇所とともに、このエラーメッセージを確認することで、エラーの原因を突き止める手がかりとする。

エラーメッセージは英語で表示されるので、英語の意味を考えればある程度想像することができる。エラーメッセージをそのまま検索にかけることで、そのエラーメッセージの意味を解説するサイトを見つけることができる場合もある。ここでは、上記のエラーメッセージの意味について解説する。

SyntaxError は「文法エラー」つまり Python の文法に則ってないということを意味している頻出のエラーである。次に、どのような文法エラーであるかが解説されている。Identifier の正確な意味[1]についてはわからなくても、無効な文字 (invalid character) と表示されていることで、「全角文字」が使われていることが原因であると推測できる。

[1] https://docs.python.org/ja/3/reference/lexical_analysis.html#identifiers

全角と半角の間違いを目で確認するのは大変だが、エラーメッセージで「エラーが発生している場所」と「無効な文字がある」という情報が示されているので、ミスを発見できるはずである。特に、全角スペースと半角スペースは見た目で区別できないので発見が大変であり、注意が必要である。

今回は日本語の文字を使っていないため、全角文字によるエラーは発生しにくいが、次回以降の授業の課題で日本語の文字を扱うようになると、日本語モードを元に戻さないことにより、半角文字で入力するべき文字を全角文字で入力してしまうことによるエラーは頻繁に出る。実際に、毎年多くの学生がこのエラーに遭遇しているので、ここでよく理解をしておくこと。

(2) ケース 2：二重引用符を全角文字にしてしまった。

print(″Hello!″)の最後の”を、全角文字の「“」として実行すると、このようなエラーが表示される。

```
  File ″Main.py″, line 1
    print(″Hello!“)
                  ^
SyntaxError: EOL while scanning string literal
```

今度もまた文法エラーである。EOL とは、End of line、つまり行が終わっている、ということであり、文字列の値 (string literal) を読み取っている途中で、行が終わってしまった、ということが書かれている。print(″ の ″で文字列の読み取りが開始して、次の″までが文字列であると解釈するところ、次の″が見つからない、というエラーが表示されていることがわかる。最後の二重引用符が「全角文字」であるために、文字列を閉じる二重引用符であると解釈されずに、エラーとなっているわけである。このように、全角文字としたことによるエラーが必ず invalid character というエラーとして表示されるとは限らないので注意すること。

(3) ケース 3：print の綴りを間違えた。

prant(″Hello!″) として実行すると、次のようにエラーが表示される。

```
NameError: name 'prant' is not defined
```

NameError は、名前のエラーである。次に 'prant' という名前は定義されていない、と表示される。ここで、prant の箇所が間違えているとわかる。そして、print の綴りを prant と間違えていることがわかる。関数の名前は大文字と小文字を区別するので、たとえば p を大文字として Print(″Hello!″)としても、

```
NameError: name 'Print' is not defined
```

のようにエラーが表示される。

【★課題】

作成した Hello! と出力するプログラムを、ToyoNet-ACE のレポートから、次の手順で提出してください。

★ Paiza のプログラムの画面とは別のブラウザのタブでToyoNet-ACEを開き、「レポート」

から、「第 3 回 Python の基礎」を選ぶ。
★ Paiza タブの Hello プログラムの画面で、プログラムのエリアにカーソルをあわせ、ブラウザのメニューから「すべてを選択」を選んで（Windows では Ctrl + A、Mac では command + A)、さらに「コピー」を選ぶことで（Windows では Ctrl + C、Mac では command + C)、プログラムをすべてクリップコードにコピーする。
★ ToyoNet-ACE のタブに戻って、テキストエリアを右クリックして「貼り付け」をする（Windows では Ctrl + V、Mac では command + V）ことで、クリップボードにコピーされているプログラムを貼り付ける。それ以外の余計な文字（スペースなど）を入れないこと。プログラムとは関係のない文字(メッセージなど)が入力されていた場合には、その文字もプログラムに実行されてエラーとなるため、不合格点となる。#で始まるコメント行や何も入力されていない空行（スペースを入力しないこと）は実行に影響を与えないので入っていても大丈夫である。
★ 教員へのコメントを書く場合には、行を##で始めること。2 行以上のコメントを書く場合には、すべての行頭に##を入れること。
★ 「プレビュー（次へ)」ボタンを押す。
★ 「状態」に赤字で「まだ提出していません」と表示されるので、さらに「提出」ボタンを押す。
★ 「状態」に「提出済み」と表示されることを確認する。
★ 提出を確認したら帰って良い。授業時間内に提出できなかった場合には、提出期限までに提出をすること。

【課題の再提出機能】
提出期限前であれば、何度でも再提出できる。「提出取消（再提出する）」ボタンを押して、さらに「戻る」ボタンを押すことで、プログラムを編集できる状態になる。

【課題の採点】
提出されたファイルを教員のパソコンで実行して、指示されている通りの結果が得られるプログラムができているかどうかで採点する。採点結果は ToyoNet-ACE で確認できる。具体的には、以下のような基準で採点する。今後の課題でも同様である。

(1) 実行時にエラーが生じるプログラムは、不合格点となる。【エラーのデバッグ】をよく確認して、エラーが出ないプログラムを提出すること。
(2) 実行時にエラーは出ないけれど想定通りの結果とならないプログラムは、プログラムの完成度によって、不合格点（60 点未満）となるか、満点ではない合格点（60 点以上）となる。今回は Hello! という文字を出力するプログラムを作る、という課題なので、他の文字が表示されるプログラム（Hallo! のような綴りの間違いを含む）は「指示通りの結果」ではないので、満点にはならない。
(3) 課題で指示された通りの動きをするプログラムが提出されていれば、満点（100 点）である（ただし、円周率の計算アルゴリズムを確認するプログラムで円周率そのものを表示するといったような「課題の趣旨を理解していないプログラム」は、不合格点となる)。「指示通りの動きをするプログラム」を作ってから提出すること。今回の課題は、書かれている通りに打ち込むだけで、指示通りの動きとなる。
(4) 未提出は 0 点であり、授業欠席となる。
(5) 提出期限（ToyoNet-ACE のレポート提出画面を確認すること）を過ぎたら、いかなる理由があっても提出・再提出は不可である。PC やネットの不調などで提出できなかった場合も同様とするため、提出期限ぎりぎりで提出しようとしないこと。
(6) 成績は 15 回の課題の平均点で決まる。

第4回　計算プログラム

入力された値に対して計算をするプログラムを作成する。

【入力を受け付けるプログラム】
★足し算をするプログラムを作成して実行しましょう。Python3 を選ぶのを忘れずに。

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

今回は「入力を受け付けるプログラム」なので、前回と同様にプログラムを打ち込んですぐに「実行」ボタンを押すと、次のようなエラーが表示される（ターミナルからプログラムのファイルを実行すると、入力待ちの状態となる）。

```
Traceback (most recent call last):
  File "Main.py", line 1, in <module>
    a = int(input())
ValueError: invalid literal for int() with base 10: ''
```

Paiza では、ユーザーの入力を「入力」欄に入れることができる。ここでは、「入力」欄に次のように2つの数を入れる。

```
23 35
```

そして「実行」ボタンを押すと、出力には入力した2つの数23と35を足した「58」が計算結果として表示される。このように、入力した2つの数を合計した数を出力するプログラムを書いたことになる。
エラーが出る場合には、前回の授業のエラーの修正方法を確認して修正すること。正常に実行されたら、入力欄に他の数（整数）を入れて試してみること。

【入力の読み取り】
このプログラムでは、input 関数によって入力を読み取っている。input() とすることで、入力から1行文字列が読み取られる。

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

の行は、現段階ですべてを理解するのは大変であるが、簡単に解説をしておく。とりあえずは雰囲気を感じ取れれば十分である。たとえば、「23 35」と入力された場合には、input() によって”23 35”という文字列が読み取られる。その後、split() によって、文字列がスペースで分割されたリストとなり、map 関数によってリストの要素にそれぞれ int 関数が適用されて文字列から整数に変換され、得られた整数が順番に a と b という変数に格納される。
まとめると、この行が実行されることで、「23 35」と入力したときに a と b という変数に23と35という整数が格納されることとなる。そのような「動き」がわかれば、現段階では十分である。変数の意味については、次に解説する。このような「入力を変数に格納する」コードは、今後も同様のコードを使う。

【変数と型】
変数は、値を入れておく箱のようなものである。数や文字などを格納することができる。Python では、変数に値が格納されるときに「データ型」[1]が決まる。ここでは３種類のデータ型について解説する。

(1) str（文字列型）前回の授業では “Hello!” という文字列を扱った。文字が並んでいるデータ型である。文字には「改行」などの特殊文字も含まれる。
(2) int（整数型）数学で定義される「整数」と同じで、1,2,3 のような正の整数、-1, -2, のような負の整数、そして 0 である。2.5 のような数は整数ではない。int 型にはメモリが 4 バイトのように決められて大きさの制限があるプログラミング言語が多いが、Python 3 の int 型はメモリの許す限りいくらでも大きな整数を扱うことができる。
(3) float（浮動小数点数型）実数を一定の精度で表現する。たとえば、2.45 のように表現される。

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

の行では、int 関数によって読み取られた文字列が整数型に変換されて、a と b という変数に格納されるため、変数 a と b はいずれも整数型の変数となる。プログラミング言語によっては、変数の型を明示的に宣言する必要があるが、Python ではこのように「整数の値が代入されたから、この変数は整数型になる」といったように自動的に変数の型が決まる。

【プログラムの解説】
(1) 2 つの整数を入力
「入力の読み取り」と「変数と型」で説明した通り、入力から 2 つの整数を読み取って、整数型の変数 a と b に格納する。同様の処理はこの授業で今後頻出する。
(2) 計算の実行
c = a + b で、a + b の計算結果が c に代入される。整数同士の足し算で結果は整数となるので、c は整数型の変数となる。数学で「=」には等式（左辺と右辺が等しい）と代入（右辺の計算結果を左辺に代入する）の意味があるが、Python では「=」は必ず代入の意味となる。したがって、右辺の a+b を計算した結果が c という変数に代入される。左辺と右辺を逆に書いてはいけない。
(3) 計算結果の表示
print 関数によって計算結果を表示する。

【基本演算】
このように、Python では足し算は「+」で計算できる。同様に、四則演算（加減乗除）とべき乗は、次のように計算できる。演算子はすべて ASCII 文字であることに注意。また、累乗は「^」ではないことにも注意。

1. 足し算：+
2. 引き算：-
3. かけ算：*
4. 割り算：/ （結果は浮動小数点数型 float）
5. 割り算の整数部：// （結果は整数型）
6. べき乗：**
7. 計算の順序は左から、「かけ算と割り算」は「足し算と引き算」よりも先に、「累乗」はそのいずれよりも先に計算される[2]。

[1] https://docs.python.org/ja/3/reference/datamodel.html#the-standard-type-hierarchy
[2] https://docs.python.org/ja/3/reference/expressions.html#operator-precedence

8. （ ）の中は先に計算される。（ ）は何重に重ねても良い。

★足し算をするプログラムを、「引き算をするプログラム」「掛け算をするプログラム」「累乗を計算するプログラム」などに変えて、いろいろな数を入れて実行しましょう。「累乗を計算プログラム」にして大きな数を入れることで、int 型には大きさの制限がないということを確認できます。

【BMI 計算プログラム】

ここまでで、「入力した数に対して計算をした結果を出力する」というプログラムを作れるようになり、これだけでかなりいろいろなプログラムを作ることができる。そこで、BMI 指数を計算するプログラムを作成して実行してみよう。

★次の BMI 計算プログラムを作成して、入力に身長(cm)と体重(kg)を「172 60」のように入れて実行しましょう。

```
a, b = map(float,input().split())
c = 10000 * b / (a ** 2)
print(c)
```

これは、BMI 指数（ボディマス指数）を計算するプログラムである。BMI 指数は、体重と身長の関係から算出した、人の肥満度を表す指数である。

身長を a[cm]、体重を b[kg]としたとき、BMI は

$$BMI = 10000\ b/a^2$$

と定義されている。日本肥満学会によると、BMI が 22 の場合が標準体重で、25 以上が肥満、18.5 未満が低体重である。BMI の定義は世界共通だが、肥満の定義は各国によって違い、男女で定義が異なる場合もある。

このプログラムを説明する。

(1) 入力を a, b という変数に格納するところは足し算プログラムと同じである。
(2) BMI 指数を計算して c に格納する。割り算 / が含まれるため float 型になる。
(3) BMI 指数を表示する。

★自分の身長、体重を入れて、BMI 指数を計算してみましょう。また、身長を変えずに、体重を増減させてみて、BMI 指数がどのように変わるか計算してみましょう。次回はこのプログラムを改良する予定です。

【★課題】

児童・生徒の肥満の程度をあらわすローレル指数は、身長を a[cm]、体重を b[kg]としたとき、以下の式で定義される。

ローレル指数 $= 10000000\ b/a^3$

べき乗の指数が２ではなくて３になっていることに注意。ローレル指数は 100 未満がやせ、130 が標準、160 以上が肥満とされる。なお、成人に対しては通常 BMI 指数が使われる。

BMI 計算プログラムの 2 行目を変えることで、ローレル指数を計算するプログラムを作成して、実行して正常に動作することを確認し、ToyoNet-ACE のレポートから提出してください。提出方法と採点基準は、前回と同様です。

ヒント：身長 140 cm、体重 36 kg のローレル指数は 131 程度です。

第 5 回　条件分岐

条件分岐の if 文について学ぶ。

【BMI 計算プログラム】
★前回作成した BMI 計算プログラムを改良して、次のプログラムを実行しましょう。最初の 2 行は前回保存したプログラムと同じなので、「一覧」「自分のコード」から前回保存したコードを呼び出すことで、続きを打ち込むことができる。今回は日本語入力を使うので、第 3 回で学習した「無効な文字」のエラーを出しやすくなるので注意すること。

```
a, b = map(float,input().split())
c = 10000 * b / (a ** 2)
if c < 18.5:
    d = "やせ型"
elif c < 25:
    d = "標準体型"
else:
    d = "肥満型"
print(f"BMIは{c:.1f}です。あなたは{d}です。")
```

実行例

入力：172 60
出力：BMI は 20.3 です。あなたは標準体型です。

あなたの身長と体重を入れて試してみましょう。

今回のプログラムでは、判定基準に基づいて、判定された結果が表示される。このプログラムの動作は、以下の様になる。

(1) 入力を受け付けて、a と b にそれぞれ身長と体重が入る。
(2) BMI 指数（身長と体重から肥満度を計算する指数）を計算し、c に格納する。ここまでは前回のプログラムと同じである。ログインして作業をしているのであれば、「一覧」「自分のコード」から前回のコードを呼び出すことができる。
(3) 赤字の箇所：BMI 指数が 18.5 未満であれば「やせ型」、18.5 以上 25 未満であれば「標準体型」、25 以上であれば「肥満型」という判定結果を変数 d に文字列として格納する。ここで条件分岐の if 文が使われているので、以下に詳しく解説する。
(4) BMI 指数と判定結果を表示する。format 関数を使って表示形式を整えているので、後に詳しく解説する。

【インデントについて】
Python では、複合文[1]をインデント（字下げ）によって表現する。たとえば

```
if c < 18.5:
    d = "やせ型"
```

[1] 複合文 https://docs.python.org/ja/3/reference/compound_stmts.html

という箇所について、if 文の意味については次に解説するが、まずは形式について着目する。複合文は1つ以上の節から成り、上の if 文は1つの節である。節は「ヘッダ」と「ブロック（スイート）」から成り、1行目の if c < 18.5:が「ヘッダ」であり、d = ″やせ型″がこのヘッダに付随するブロックである。節に対するブロックはインデントが下がり、ブロックが複数の行にわたるときには同じレベルのインデントとなる。

Python では、このようにインデントがプログラムを解釈するための重要な意味を持つため、不要なインデントを入れてしまうとエラーとなる。インデントは「スペース」または「タブ」によって作られ、その数は同じブロック内で等しければいくつであっても良いが、「4個のスペース」を使うことが標準として推奨されている(「全角スペース」は使わないこと)。Paiza では、Tab キーを押すことで4個のスペースが入力される。また、if 文のステートメントブロックを入れて Enter キーを押すと、自動的にインデントがされる。さらには、インデントされた行で改行をすると、自動的にインデントが継続する。インデントを元に戻すにはバックスペース(BS)キーを使う。

【基本的な if 文】

if 文はプログラミング言語が最低限備えている必要がある制御構造である「選択」「ループ」の中で「選択」という制御構造である。

基本的な if 文の構造

```
if 条件式:
    条件が成り立つときの処理
```

「条件が成り立つときの処理」は、同じインデントブロックであれば何行でも記述できる。たとえば、次のように記述する。

```
if 条件式:
    条件が成り立つときの処理1
    条件が成り立つときの処理2
    条件が成り立つときの処理3
```

条件式の値は、条件を満たすときに True、満たさないときに False となる。条件式の値は、必ずこのどちらかになる。式の型は bool 型である。

たとえば、c < 18.5 の時に d に「やせ型」という文字列を代入する処理は、次のようになる。

```
if c < 18.5:
    d = ″やせ型″
```

ここで、if c < 18.5: という文の中で、c < 18.5 が条件式である。条件式 c < 18.5 は、c の値が 18.5 未満の時に True となり、18.5 以上の時に False となる。したがって、c<18.5 の時に d = ″やせ型″が実行され、そうでない場合には実行されない。

数の大小比較をする比較演算子をまとめる。

Python	数学	意味
c == 18.5	c = 18.5	c が 18.5 に等しいなら True、等しくなければ False
c!= 18.5	c ≠18.5	c が 18.5 に等しくないなら True、等しければ False

c >= 18.5	c ≧ 18.5	c が 18.5 以上なら True、それ以外なら False
c <= 18.5	c ≦ 18.5	c が 18.5 以下なら True、それ以外なら False
c > 18.5	c > 18.5	c が 18.5 より大きいなら True、それ以外なら False
c < 18.5	c < 18.5	c が 18.5 より小さいなら True、それ以外なら False

ここで、「等しい」を意味する比較演算子は、= を 2 個重ねていることに注意する。前回説明したように、Python では = は代入と解釈されるため、「等しい」を意味する場合には == と重ねることで区別する。if 文で == と書くべきところを = と書いてしまう間違いは、わかっていてもうっかりと間違いやすいところである。

【if … else 文】
次に、条件式が True の場合と False の場合、それぞれ別の処理を書く if … else 文の構造を示す。

```
if 条件式:
    条件が成り立つ時の処理
else:
    条件が成り立たなかった時の処理
```

たとえば、c < 18.5 の時に d = ″やせ型″、そうでない時に d = ″やせ型ではない″を実行するには、次のように書く。

```
if c < 18.5:
    d = ″やせ型″
else:
    d = ″やせ型ではない″
```

【elif による if 文の連鎖】
else と if を連結した elif を使って、次のように if 文を連鎖することができる。

```
if 条件式１:
   条件式１を満たすとき
elif 条件式２:
   条件式１は満たさず、条件式２を満たすとき
else:
   条件式１は満たさず、条件式２も満たさないとき
```

冒頭のプログラムの赤字の箇所では、この構造を使って、c の大きさによって 3 通りに d の値を変化させる処理を実現している。

【f 文字列による表示形式の整え方】
プログラムの最後の行の

```
print(f″BMIは{c:.1f}です。あなたは{d}です。″)
```

について解説する。print 文の引数には、

f″BMI は{c:.1f}です。あなたは{d}です。″

と書かれているが、これはｆ文字列（f-strings、フォーマット済み文字列リテラル）である。ｆ文字列は先頭に「f（あるいはF)」がついている文字列であり、波括弧{ }で囲まれた「置換フィールド」の内容が式として評価されて表示される。たとえば、{c}と書くことでｃが式として評価され、変数ｃの値に変換される。さらに、: に続いて書式指定文字列を追加できる。たとえば、数値の小数点以下１桁までを表示する書式指定文字列は.1f である。したがって、{c:.1f}と書くことで、変数ｃの値を小数点以下１桁まで表示することとなる。

ｆ文字列について詳しくは
https://docs.python.org/ja/3/reference/lexical_analysis.html#formatted-string-literals

書式指定文字列について詳しくは
https://docs.python.org/ja/3/library/string.html#formatstrings

【最大値を求めるプログラム】
★3つの整数の最大値を計算するプログラムを作成して実行をしましょう。

```
a, b, c = map(int,input().split())
m = a
if b > m:
    m = b
if c > m:
    m = c
print(m)
```

実行例

入力：4 5 7
出力：7

【★課題】
上記プログラムの赤字の箇所を修正して、3つの整数値の「最小値」を出力するプログラムを作成して、実行をして動作を確認してから、ToyoNet ACE経由で提出して下さい。赤字以外の場所は書き換えないこと。

実行例

入力：4 5 7
出力：4

入力：13 2 5
出力：2

【発展：Pythonの禅】
「発展」では学生が自主的により深い学習をするためのきっかけとなるトピックを取り上げる。次のプログラム（イースターエッグ）を実行してみよう。

```
import this
```

第 6 回　繰り返し処理

前回は、条件分岐の if 文について学んだ。今回は、繰り返し処理について学ぶ[1]。

【for 文を使った繰返し処理】
for 文を使った繰り返し処理の構造を示す。

```
for 変数名 in イテラブルオブジェクト:
    繰り返す処理
```

ここで「イテラブルオブジェクト」とは、「繰り返し可能なオブジェクト」ということである。繰り返す(iterate)ことが可能という意味の形容詞が iterable である。「イテラブルオブジェクト 」という言葉は少し難しいので、具体例で考える。

for 文の例（0,1,2,3,4 を順番に表示する）

```
for i in range(5):
    print(i)
```

ここで使われている range() という関数が、連続した数値の「イテラブルオブジェクト」を生成する関数である。整数 n に対して range(n)とすることで、0 から n-1 までの順番に連続した整数を生成する。したがって、range(5)では 0, 1, 2, 3, 4 という順番に連続した整数が生成される。この連続した整数を、順番に i という変数に代入して「繰り返す処理」である print(i)を実行する。ここで、range(5)は 0 から 4 までの整数なので、全部で 5 個あり、繰り返しは 5 回実行される。
range(n)は 0 から n-1 までの数を返すが、range(a,b)とすれば a から b-1 までの数を返す。また、range(a,b,c)とすると、a から b に届く前までを c おきに返す。

例：range(2,5) ： 2,3,4
　　range(2,7,2) ： 2,4,6
　　range(2,8,2) ： 2,4,6
　　range(8,2,-1) ： 8,7,6,5,4,3
　　range(8,2,-2) ： 8,6,4
　　range(8,3,-2) ： 8,6,4

★次のプログラムを実行してみましょう。入力には整数（たとえば 10）を 1 つ入れます。

```
n = int(input())
sum = 0
for i in range(1,n+1):
    sum += i
print(sum)
```

[1] コンピュータの制御構文（プログラムの実行順序を制御する方法）の中で、最も基本的なものが条件分岐（選択）と繰り返し（ループ）である。条件分岐と繰り返しがあれば、あらゆるプログラムの構造を書き換えることができ、条件分岐は繰り返しで書き換え可能であるため実は繰り返しだけでも良い。

これは、何をするプログラムでしょうか？プログラムと実行結果を見て、考えてみよう。

【while 文を使った繰返し処理】

繰り返し処理をする方法には、for 文だけでなく、while 文を使う方法がある。そこで、次に while 文について学ぶ。

while 文の構造

```
while 条件式:
    繰り返す処理
```

「条件式」は、繰り返しを続ける条件を示した式である。ここに書いた式が満たされている間に限り（条件式の評価結果が True になっている間に限り）、繰り返しの処理が実行される。前回学習した、if 文の構造と見比べてみよう。

```
if 条件式:
    条件が成り立つ時の処理
```

while と if の比較

（共通点）while 文も、if 文も「条件式」の判断をして、その値が True になっている間に限り処理が実行される。

（相違点）while 文では、条件式が成り立つ間、何回でも処理が繰り返される。if 文では、処理が実行されるのが 1 回だけ。

while 文の例（0,1,2,3,4 を表示する）

```
i = 0
while i < 5:
    print(i)
    i += 1
```

1. i = 0 で、i に整数 0 が代入される。
2. 条件式(i<5)を評価する。正しい(True)ので、繰り返し処理が実行され、i の値 0 が表示される。
3. i += 1 が実行される。これは「変数 i の値を 1 増やす」という意味で、i の値が 1 になる。
4. while 文に戻って条件式(i<5)を評価する。正しい(True)ので、繰り返し処理が実行され、i の値 1 が表示される。
5. i += 1 が実行される。i の値が 2 になる。
6. while 文に戻って条件式(i<5)を評価する。正しい(True)ので、繰り返し処理が実行され、i の値 2 が表示される。
7. i += 1 が実行される。i の値が 3 になる。
8. while 文に戻って条件式(i<5)を評価する。正しい(True)ので、繰り返し処理が実行され、i の値 3 が表示される。
9. i += 1 が実行される。i の値が 4 になる。
10. while 文に戻って条件式(i<5)を評価する。正しい(True)ので、繰り返し処理が実行され、i の値 4 が表示される。
11. i += 1 が実行される。i の値が 5 になる。

12. while 文に戻って条件式(i<5)を評価する。誤っている(False)ので、繰り返し処理は実行されず、while 文が終了する。

★次のプログラムが、先ほどのプログラムと同じように動作することを確認してみよう。

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

【複合文のネスト】

if, for, while などの複合文は、複合文のブロックの中に別の複合文を書くことで、多重の入れ子構造にすることができる。このような入れ子構造のことを「ネスト」と言う。for, while 文をネストすることで、多重ループによる繰り返し処理ができる。

★次の掛け算九九を表示するプログラムを実行してみましょう。

```
for i in range(1,10):
    for j in range(1,10):
        print(f"{i} * {j} = {i*j}")
```

【ループを途中で抜ける】

ループを途中で抜けるには、break を使う。

```
for i in range(10):
    if i == 3:
        break
    print(i)
```

【三角形を表示するプログラム】

★次の左下が直角の三角形を表示するプログラムを実行してみましょう。

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

入力：5
出力：*
**
***
****
*****

ここで、print ('*'*i) という文では、文字列の繰り返し演算が使われている。

“文字列” * 回数

と書くことで、同じ文字列を指定回数だけ繰り返した文字列となる。

【★課題】
直前のプログラムの赤字の箇所を修正して，左上が直角の三角形を表示するプログラムを作成して､そのプログラムをToyoNet-Aceから提出して下さい｡3以外の入力も試すこと。

入力：3
出力：***
　　**
　　*

【発展：ドキュメントと外部モジュール】
Pythonのプログラミングでわからないところは、まずは公式のドキュメントを確認する。公式のドキュメントは、かなりの部分が日本語に訳されている[1]ので、この授業でPythonの扱い方に慣れてきたら、まずは「チュートリアル」を一通り読むことから始めると良いであろう。具体的な関数の使い方などについて詳しく知りたいときには、「ライブラリーリファレンス」「言語リファレンス」から該当する説明を探すのが良い。この授業でこれまでに解説してきた事項も、これらのドキュメントの該当する箇所を読むことで、さらに理解が深まるであろう。あるいは、ドキュメントを読んで理解できるようになることが、この授業の目標到達点であるとも言える。
Python は標準ライブラリだけでもかなり便利に使うことができるが、外部モジュールを読み込むことでさらに便利に使うことができる。外部モジュールは、まずはシステムにインストールしてから、プログラム内で import して使う。モジュールのインストールには、pip というシステムを使う。詳しくは「Python モジュールのインストール」を参照。Paiza を使う場合には、Paiza にインストールされているモジュールでなければ直接 import して使うことはできない。Python は世界中で多くの人が使っているため、質の高いモジュールが数多くあり、多くの人たちによってメンテナンスされている。せっかく Python を使うのであれば、そのようなモジュールをうまく活用して効率良くプログラミングすることでそのメリットを享受できる。そのような外部モジュールについては、詳しくはモジュールのドキュメントを確認することになるので、ドキュメントを探して読み解くことのできる能力は、Python プログラミングにおいてとても重要である。
公式のドキュメントを読んでも十分にわからないことは、インターネットで検索することで解決することもある。なるべく、知りたいことを具体的にしぼってキーワードで検索すると良い。たとえば、エラーメッセージがある場合には、エラーメッセージをそのまま入れると良い検索結果が得られることがある。特に Python は世界中に多くのユーザーがいるため、日本語で検索しても十分に良い検索結果が得られなくても、英語で検索すると良い結果が得られることがある。英語に自信がない人も、最近は機械翻訳の精度が上がっているため、機械翻訳を使えば十分に理解ができるであろう。ただし、いきなりネットで検索してもあまり質の高い結果が得られない場合があるので、まずは公式ドキュメントを読んで関連する基本事項をよく理解することが重要である。

[1] https://docs.python.org/ja/3/

第 7 回　コンテナのデータ構造

データの集まりをまとめて扱うコンテナのデータ構造について学ぶ。

【コンテナのデータ構造】
複数のオブジェクト（データ）をひとまとめに格納するデータ構造をコンテナと呼ぶ。複数のデータの集まりであることからコレクションとも呼ばれる。いずれも様々なデータ構造を含む包括的な名称である。Python では、次のような組み込みのコンテナのデータ型が用意されている。

型の名前	表記例	主な特徴
リスト型 list	[1, 2, 3]	順番に並んでいる。要素を変更できる。
タプル型 tuple	(1, 2, 3)	順番に並んでいる。要素を変更できない。
集合型 set	{1, 2, 3}	順序は関係ない。要素は重複しない。
辞書型 dict	{"one":1,"two":2}	順序は関係ない。キーで要素を呼び出す。

array モジュールや numpy モジュールを読み込むことで配列(array)が使われることもある。一般に、配列ではすべての要素のデータ型が同じである必要があるが、リストやタプルの要素は同じデータ型である必要はない。リストの要素がコンテナであっても良く、リストの中にリストが入ったものを多重リスト（多次元リスト）と呼ぶ。
これまでにも扱ってきた文字列型 str は、文字が順番に並んだデータ型であり、文字列はリストと同じように扱うことができる（ただし、要素の変更はできないのでタプルに近い）。リスト型とタプル型、文字列型、そして前回の授業で扱った range 関数が生成する range 型のようにデータが一列に順番に並んだデータ型を「シーケンス型」と呼ぶ。

【リスト】
この授業では、シーケンス型の中でもリストを中心に扱う。リスト型のデータは、複数のデータを順番に並べたものである。たとえば、5, 2, 9, 7, 1 という 5 個の整数を順番に並べたリストは、[5, 2, 9, 7, 1]のように表記する。リストの要素は同じデータ型である必要はなく、たとえば、[1, 2, “りんご”]のように、整数型と文字列型が混在するリストを作ることも可能である。さらには、[1, 2, [1, 3, 4], (1, 2, 3)]のように、リストの中にリストやタプルを入れることも可能である。
リストに含まれる個々のデータをリストの「要素」、リストに含まれる要素の個数をリストの「長さ」と言う。
リストは「順番に」並べられているので、リストの要素は「何番目の要素か」ということによって指定することができる。これをリストの「インデックス」と言う。インデックスは、リストの何番目にデータが格納されているかを表す数値で、1 からではなく 0 から数える。
リストの要素を参照するには、リストの名前[インデックス] と指定する。list という名称のリストがあった場合、この 3 番目(1 から数えると 4 番目)の要素を参照するには list[3] と指定する。たとえば、

```
list = [5,2,9,7,1]
print(list[3])
```

というプログラムを実行すると、「7」と出力される。「9」ではないことに注意。

インデックスには変数を用いることができる。list の i 番目の要素を参照するには list[i]と指定する。この場合には、i=1 であれば list[1]が、i=3 であれば list[3]が参照される。

★次のプログラムを実行してみましょう。

```
list = [5,2,9,7,1]
print(f"リストの長さは{len(list)}です。")
for i in range(len(list)):
    print(f"{i}番目の要素は{list[i]}です。")
```

これまでの授業を思い出しながら、このプログラムを一行ずつ読んで、どのようにプログラムが実行されているか考えてみましょう。for 文による繰り返し処理で、添字 i を用いてリストの要素を参照していることを確認して下さい。

リストは要素を変更することができるため、たとえば list[2] = 3 とすれば、2 番目の要素（0 から数えるので実際には 3 番目の要素）が 3 になる。

★上のプログラムで、1 行目を「list = "python"」のように好きな文字列に変えて実行してみてください。「文字列はリストと同じように扱える」ということの意味が分かります。

★リストや文字列を for 文の「イテラブルオブジェクト」として使うことができます。次のプログラムを実行してみましょう。

```
for c in "月火水木金土日":
    print(c + "曜日")
```

【リストの要素の最大値を求める】

リストの要素の中から最大値を求めるプログラムを考える（最大値を計算するメソッドを使えば良いが、ここでは自分でそのようなプログラムを作る方法について考える）。第 5 回目の授業で作成したプログラムでは、３つの変数 a,b,c の大きさを順番に比べていた。変数の数が多くなると、大小の比較を１つ１つプログラムに書いていくのは大変である。たとえば、a,b,c,d,e の最大値を比べるプログラムを書くと、どうなるだろうか。

そこで、リストを使うと、プログラムをより効率的に書くことができる。また、リストの要素数が決まっていない場合にも、プログラムを発展させることが可能となる。

★次の最大値を求めるプログラムを実行してみましょう。

```
list = list(map(int,input().split()))
max = list[0]
for i in range(len(list)):
    if list[i] > max:
        max = list[i]
print(max)
```

入力：5 2 9 7 1
出力：9

1 行目では、5 2 9 7 1 のようにスペースで整数を区切って入力されたものが、整数のリスト[5, 2, 9, 7, 1]に変換されて、リスト型変数 list に代入される。その仕組みは若干複雑であるが、以下に説明する。このような処理を簡潔に書けるところは Python の魅力であるが、ここは理解できなくても当面は問題ない。入力として「5 2 9 7 1」のような文字列が与えられたときに、これを input()で受け取って split()でスペースの区切りごとに分けられたリストにする（['5', '2', '9', '7', '1']のようなリストとなる）。そして、map(int,input().split()) によって要素がそれぞれ文字列から整数型に変換される。ここまではこれまでと同じ処理で、これまでは a, b = map(int,input().split()) のような形で変換された整数をそれぞれ順番に変数に格納していたが、ここでは変換されたデータをリスト型の変数として list にそのまま格納するために、map オブジェクトをリスト型に変換するための list 関数を使っている。

【リストの要素を逆順に並べ替える】

リストの要素を逆順に並べ替えるプログラムを考える。長さが 5 のリストの場合、

0 番目の要素と 4 番目の要素を交換する
1 番目の要素と 3 番目の要素を交換する

という処理を実行すれば、配列の要素を逆順に並べ替えられる。

ここで、0 番目の要素と 4 番目の要素を交換する場合に、

```
list[0] = list[4]
list[4] = list[0]
```

とすると、どのような問題が起きるだろうか？

上記の方法では問題が起きるので、交換対象のデータを一時的に保持する変数 tmp を用意し、

```
tmp = list[0];
list[0] = list[4]
list[4] = tmp
```

とするのが、一般的なアルゴリズムである。Python では、このような一時的な変数を使わずに

```
list[4], list[0] = list[4], list[0]
```

とまとめて書くことができる。

【★課題】

次のプログラムは、リストを逆順に書き換えるプログラムを作ろうとして「失敗した」ものです。まずはこの通りに実行して、どのような結果になるのかを確認してください。その後、このプログラムの赤字の箇所を書き換えて、正しく逆順になるようにしてから、ToyoNet-ACE から提出して下さい。赤字の箇所以外は書き換えないこと。

```
list = list(map(int,input().split()))
for i in range(len(list)):
    j = len(list) - 1 - i
    list[i], list[j] = list[j], list[i]
print(list)
```

【課題のヒント】

上記のプログラムは、リストの長さが 5 のときには、次の処理を実行している。

0 番目の要素と 4 番目の要素を交換する
1 番目の要素と 3 番目の要素を交換する

2 番目の要素と 2 番目の要素を交換する
3 番目の要素と 1 番目の要素を交換する
4 番目の要素と 0 番目の要素を交換する

すると、結局元に戻ってしまう。この交換を、2 番目あるいは 3 番目までで「止めて」しまえば、うまくいくはずである。そのためには、赤字の range の中を変えてどこまで繰り返しを進めるかを変えれば良い。len(list)が 5 のときには 2 あるいは 3 となり、len(list)が 10 であれば 5 となるような計算にする。どのような計算（演算）を使えば良いかは、第 4 回の授業の「基本演算」を読み、+, -, *, // の中から選んで使うこと。ここで、range 関数は整数型のみ許容されるので、float 型が返る / を使うとエラーになることに注意すること。

正しく書き換えると、次のような計算結果となる。

入力：5 2 9 7 1
出力：[1, 7, 9, 2, 5]

入力：1 2 3 4 5 6 7 8
出力：[8, 7, 6, 5, 4, 3, 2, 1]

【発展：リスト内包表記】
新しいリストを作るときに「リスト内包表記」を使うと便利である。次のプログラムを実行してみよう。

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

map 関数と無名関数（ラムダ式）を使って次のように書いても同じである。

```
print(list(map(lambda x: 10**x,range(10))))
```

【発展：Python でデータ解析】
Python でデータ解析をするときによく使われるライブラリが Pandas と Matplotlib である。Pandas は大きな表データを扱うときに便利で、Excel のような表計算ソフトのかわりとして使うことができる。特に、様々な形式のデータを入力として受け取り、細かいデータの加工をして統計的に集計するときには、Pandas が便利である。また、Pandas でデータの基本処理をしてから、数値計算ライブラリの NumPy で線形代数などのさらに高度な数値計算をすることもある。
Matplotlib は、グラフを表示するためのライブラリである。Pandas は Matplotlib と組み合わせて使われることが多く、Pandas でデータを読み込んで計算をして、それを Matplotlib で表示する、というところまでを自動化することで、定型的なデータ集計作業をプログラミングすることが可能である。グラフの表示は様々なタイプがあり、細かく制御できる。Matplotlib のページのギャラリー(https://matplotlib.org/stable/gallery/)に、様々なグラフとそれをどのように描画するかの例が示されている。

第 8 回　再帰的アルゴリズム

再帰的アルゴリズムについて学ぶ。

【関数の定義】
これまでは、print()や range()のように Python に組み込まれている関数を利用してきたが、関数を自作することができる。たとえば、第 4 回の授業で作成した身長と体重から BMI を計算する次のプログラムを、関数を自作することで書き換えてみよう。

```
a, b = map(float,input().split())
c = 10000 * b / (a ** 2)
print(c)
```

このプログラムは、関数 bmi を定義することで次のように書き変えることができる。

```
def bmi(a,b):
    return 10000 * b / (a ** 2)

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

1 行目の def bmi(a,b): からはじまるブロックが、関数の定義である。2 行目までがインデントされていることで、2 行目までが関数の定義であることがわかる。def は関数を「定義(define)しますよ」ということであり、bmi は関数の名前である。そして、(a,b)では関数には 2 つの引数があり、それぞれを a と b という引数として受け取る。つまり、def bmi(a,b)は bmi という名前の関数には引数 a と b がある、ということを意味する。
2 行目の return とは、関数を終了することを意味する。return とだけ書かれると、戻り値がなく、return の後に書かれているものを「戻り値」として返す。つまり、

```
def bmi(a,b):
    return 10000 * b / (a ** 2)
```

の 2 行で、a と b を引数として受け取り、10000 * b / (a ** 2)の計算結果を戻り値として返す bmi という名前の関数が定義された。
その後の 3 行では、入力を a と b という変数で受け取り、この bmi 関数を使って計算した結果を出力する。
つまり、このプログラムは最初の BMI 計算プログラムとまったく同じ動きをする。bmi 関数は 1 回しか使われていないので、ここでは関数を定義するメリットはそれほど感じないが、同じ関数を何回か使うときには便利であり、関数に名前をつけて整理することでプログラムがわかりやすくなる。

【再帰的呼び出し】
関数の再帰的呼び出し(recursive call)とは、ある関数から自分自身を呼び出す処理である。例として、階乗 n!（n は非負整数）を考える。n! は次の定義で得られる。

(1) 0! = 1
(2) n! = n × (n - 1)!　　(ただし n > 0)

例えば5!は、5 × 4! と計算される。つまり、5! を求めるという計算の中で4!を求めるという関数を再帰的に呼び出すこととなる。4!は 4 × 3! なので、さらに再帰的に階乗を求める関数が呼び出される。これが 0! に到達するとようやく 1 という数値が得られ、結果として，5! = 5 × 4 × 3 × 2 × 1 になる。このように、自分自身を呼び出すアルゴリズムを再帰的アルゴリズムと言う。

再帰的定義とは、定義の中に、更にその定義されるべきものが簡単化されて含まれていることである。まったく同じ定義が含まれるときには、循環論法といって定義が定まらない。再帰的とはその部分に含まれるものが全く同じものではなくて、簡単になったもので、最終的終了条件が明示されているものである。

★階乗を求めるプログラムを実行してみよう。

```
def factorial(n):
    if n > 0:
        return n * factorial(n-1)
    else:
        return 1

n = int(input())
print(factorial(n))
```

実行例

入力：5
出力：120

入力：50
出力：30414093201713378043612608166064768844377641568960512000000000000

★この階乗の関数を利用して、ネイピア数（自然対数の底）の近似計算をしてみよう。

```
def factorial(n):
    if n > 0:
        return n * factorial(n-1)
    else:
        return 1

import math
e = 0
for k in range(20):
    e += 1 / factorial(k)
    print("k={0}: {1}".format(k,e))
print(f"  e = {math.e}")
```

ネイピア数は次の極限となることを使っている。

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots$$

【ハノイの塔】

再帰アルゴリズムを用いて、ハノイの塔を解くプログラムを作成する。ハノイの塔とは、3 本ある杭を用いて，以下のルールに従ってすべての円盤を右端の杭に移動させられれば完成、というゲームである。

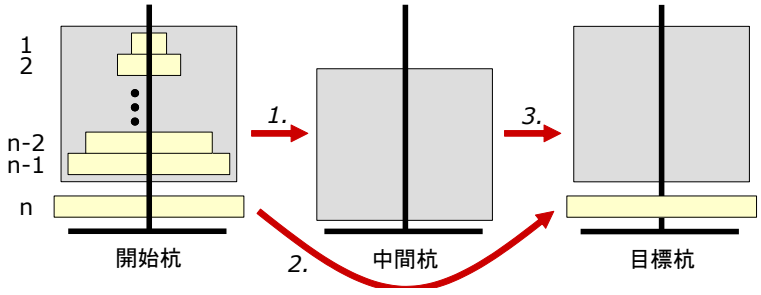
(1) 3 本の杭と、中央に穴の開いた大きさの異なる複数の円盤から構成される。
(2) 最初はすべての円盤が左端の杭に小さいものが上になるように順に積み重ねられている。
(3) 円盤を一回に一枚ずつどれかの杭に移動させることができるが、小さな円盤の上に大きな円盤を乗せることはできない。

https://hanoi.aimary.com/ で、ハノイの塔を実際にやってみよう。

この問題は、次のように考えれば解決できる。n 枚の円盤を開始杭から目標杭へと移動させる問題を、(n-1)枚の円盤のグループと、n 枚目の円盤 1 枚に分ける。そして、

(1) (n-1)枚の円盤のグループを開始杭から中間杭に移動させる
(2) n 枚目の円盤を開始杭から目標杭に移動させる
(3) (n-1)枚の円盤のグループを中間杭から目標杭に移動させる

という問題としてとらえる。



このように考えると、次に考えるべき問題は、(n-1)枚の円盤のグループを、開始杭から中間杭にどうやって移動させるか？という問題になる。つまり、n 枚の円盤を移動させる問題から、n-1 枚の円盤を移動させる問題へと、問題の規模が縮小したということである。これが再帰的アルゴリズムである。

【★課題】
ハノイの塔を実行する次のプログラムの????を埋めて完成させ、実行してみよう。正常に実行されたら、ToyoNet-ACE に提出してください。# で始まる行はコメントであり入力してもしなくても構いません。

```
def move(n, start, end):
    # n-1枚の円盤を start 番目の杭から中間杭へと移動する
    if n > 1:
        move(n-1, start, ????)
    # n 枚目の円盤を移動（移動結果を出力）
    print(f"円盤{n}を杭{start}から杭{end}へと移動")
    # n-1 枚の円盤を中間杭から end 番目の杭へと移動する
```

```
    if n > 1:
        move(n-1, ????, ????)

n = int(input())
move(n,1,3)
```

関数 move には,3 つのパラメータが必要である。第 1 パラメータが移動する円盤の枚数、第 2 パラメータが移動元（開始杭）の杭番号、第 3 パラメータが移動先（目標杭）の杭番号である。杭には 1 から 3 までの番号がついていて、はじめは開始杭の番号が 1 で目標杭の番号が 3 であるが、再帰的な呼び出しをするときには、開始杭と目標杭の番号が変わるために、それぞれをパラメータとして関数を呼び出し、move 関数の中では start と end という変数に格納される。

このプログラムのポイントは，「中間杭の番号を start と end の組み合わせでどう表現するか？」である。杭の番号は 1，2，3 で、これらの合計は 6 なので、もう分かるであろう。例えば 2 が中間杭だとすると，2 = 6 − 1 − 3，すなわち，中間杭 ＝ 6 − 開始杭 − 目標杭で求められる。あとは、これをプログラム中の変数を使った式で表現すれば良い。

【発展：NumPy と線形代数】

NumPy（ナンパイ）は、科学技術計算でよく利用される数値計算ライブラリである。NumPy の配列は、ベクトル、行列、テンソルなどの線形代数の計算を高速に実行できるところがその強みである。特に、最近の流行である人工知能（AI）の機械学習は、線形代数がその「すべて」であると言っても過言ではなく、線形代数の理解なしには語れない。Google が提供している機械学習ライブラリ TensorFlow は、その名の通り要するにテンソルの計算を高速にするためのライブラリである。Google Colaboratory[1]（略称: Colab）を使うと、Google のクラウドサーバー上で Python と TensorFlow を使って計算をすることができる。

近年は高校の数学でベクトルや行列を学ぶ機会がなくなっているため（ベクトルが教えられる「数学 B」は履修者が少ない）、線形代数になじみのない学生が多いであろう。この授業では線形代数や NumPy について深入りはしないが、以下のサンプルコードを動かしてみよう。

https://paiza.io/projects/AP4Xuxa5ZQ9Ndxxnmz48bw

```
import numpy as np
a = np.random.randint(0, 10, size=(3,3))
print('行列A')
print(a)
print('Aの転置行列')
print(np.transpose(a))
print('Aの逆行列')
print(np.linalg.inv(a))
w, v = np.linalg.eig(a)
print('Aの固有値')
print(w)
print('Aの固有ベクトル')
print(v)
```

[1] https://colab.research.google.com/

第 9 回　整列アルゴリズム（バブルソート）

今回と次回は整列アルゴリズムについて学ぶ。

【単純交換ソート（バブルソート）】
リストの要素を昇順（または降順）に並べ替えるためのもっとも単純な方法として、単純交換ソート（またはバブルソート）と呼ばれる方法がある。
単純交換ソートでは、リストの後ろ側から二つの要素を比較して、先頭側が大きければ交換するという操作を、先頭の要素に到達するまで繰り返す。この繰り返し処理を一回実施すると、リスト中の最小値が先頭に置かれ、リストの先頭が「整列済み」となる。同じ処理を「整列済み」でない要素に対して繰り返す。この処理を、リスト中のすべての要素が「整列済み」となるまで繰り返す。小さな値が水底から浮かび上がってくるように見えることから、バブルソートとも呼ばれる。

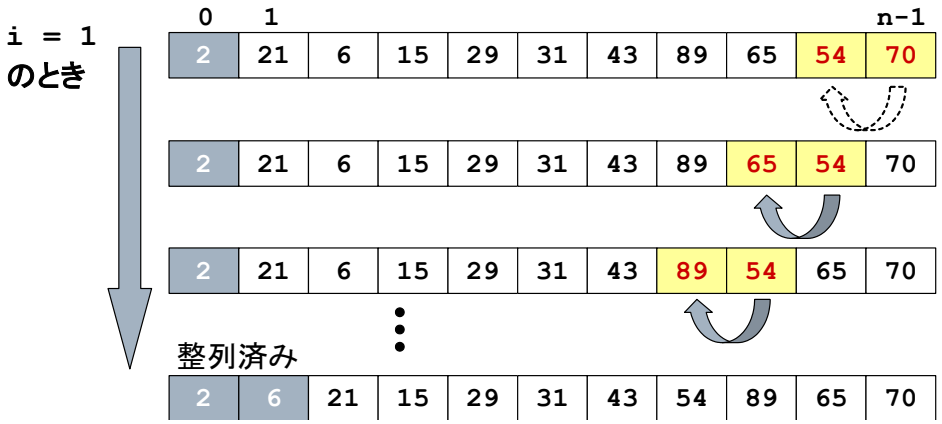
単純交換ソートの繰り返し部分は、次のようになる。

```
for i in range(len(list)):
    list[i], list[i+1], ..., list[n-1] に対して，後ろ側から二つの要素を
    比較して，先頭側が大きければ交換する；
```

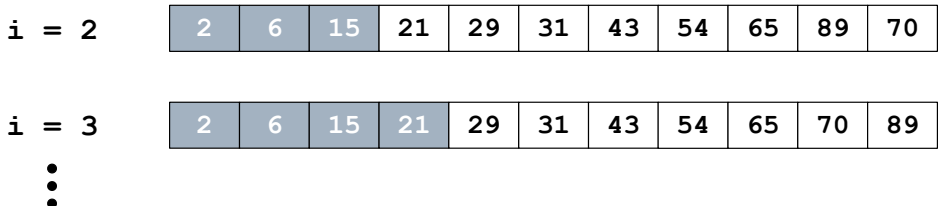
図で表すと、このようになる。

まず，繰り返し処理の 1 回目では，最初に n-1 番目と n-2 番目の要素を比較し、n-2 番目の要素の方が大きい値であればこれらを交換する。次に n-2 番目と n-3 番目を比較、n-3 番目と n-4 番目を比較、・・・、最後に 0 番目と 1 番目を比較して、先頭の要素のみ整列済みとなる。

続いて繰り返し処理の 2 回目である。n-1 番目から順に 1 番目の要素までを並べ替える。



同様に繰り返し処理を進めていけば、先頭から順に「整列済み」になる。



★次の単純交換ソートのプログラムを実行してみましょう。

```
list = list(map(int,input().split()))
for i in range(len(list)):
    for j in range(len(list) - 1, i, -1):
        if list[j-1] > list[j]:
            list[j-1], list[j] = list[j], list[j-1]
print(list)
```

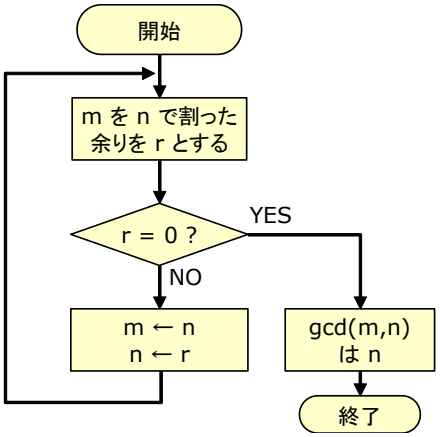
入力：21 6 15 2 29 31 89 65 43 70 54
出力：[2, 6, 15, 21, 29, 31, 43, 54, 65, 70, 89]

入力：21 6 15 2 29
出力：[2, 6, 15, 21, 29]

【ユークリッドの互除法：最大公約数を求めるアルゴリズム】

ソートアルゴリズムについては、次回の「クイックソート」でさらに詳しく学ぶが、今回は前回の授業で学んだ「再帰的アルゴリズム」の例として、ユークリッドの互除法を学習し、そのプログラムを完成させることを課題とする。

ユークリッドの互除法は、最大公約数を再帰的処理によって求めるアルゴリズムである。紀元前 300 年頃に記されたユークリッドの「原論」に書かれてあり、今日も使われている最古のしっかりとしたアルゴリズムとして知られる。フローチャートを以下に示す。



【★課題】
以下のユークリッドの互除法により最大公約数を求めるプログラムについて、????の部分を書き換えて完成させ、正しく実行されることを確認してから、ToyoNet-ACE から提出して下さい。????以外の箇所は変えないこと。

```
def gcd(m, n):
    r = m % n
    if r == ????:
        return ????
    else:
        return gcd(????, ????)

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

入力：42 28
出力：14

入力：7 21
出力：7

【剰余】
剰余とは、割り算をしたときの余りである。整数 a を正の整数 n で割り算をして、商 q 余り r になるということは、
a = nq + r (0 ≦ r < n)
となるような q と r を求めるということである。
Python では、% が剰余を計算する演算子である。上の式において、a % n という演算で剰余 r が計算される。

【計算の流れ】
例えば，m=42，n=28 が与えられたとする。
① 42 を 28 で割った余り 14 が r に入る。
② r はゼロではないので、else 節に入り，gcd(28, 14) となるよう再帰呼び出しをする。
③ 28 を 14 で割った余り 0 が r に入る。
④ r はゼロなので、最大公約数 14 が得られる。
⑤ 以上をまとめると gcd(42,28) = gcd(28,14) = gcd(14,0) のように計算される。

【再帰エラー】

今回は、RecursionError: maximum recursion depth exceeded in comparison という実行時エラーが生じる可能性がある。これは、最大の再帰の深さを超えてしまったということである。

前回学習した階乗のプログラムでは、 factorial(n) のメソッドの中で factorial(n-1) のように、引数を n から n-1 へと減らして呼び出すことで、引数が 5,4,3,2,1 のように減っていって処理が終了したが、もし factorial(n) の中で factorial(n) を呼び出せば、100 万回自分自身を呼び出しても、プログラムが終了しない。コンピュータの資源は有限なので、最大の再帰深さが設定されていて、それを超えたらエラーが出るように設計されているため、このエラーが生じる。

つまり、この実行時エラーが出る場合には、再帰呼び出しの引数がおかしいであろうということが想定できるので、なぜそうなるのかよく考えてみること。

【発展：GUI アプリケーションの作成】

この授業で扱うのは、Paiza によって「入力」をテキストで与えたときに「出力」がテキストで返るようなテキスト入出力のプログラムである。画像を表示したり、マウスで操作をしたりする GUI（グラフィカルユーザーインターフェース）のアプリケーションを作るためにはどうすれば良いであろうか。ここでは、2 つ紹介する。

(1) Tkinter

Tkinter は、Python から GUI を構築・操作するための標準ライブラリである。標準ライブラリなので別途インストールする必要はない。スクリプト言語 Tcl/Tk（ティクル・ティーケー）の Tk を使うインターフェイスということで、Tkinter という名称となっている。

Tkinter を使ったプログラミングの例として、すがや(2020)の Python 入門漫画で解説されているマウスを使ったスカッシュゲームを紹介する。サポートページ[1]からダウンロードできる。『ゲームセンターあらし』は、私が小学生のときによく読んでいた「コロコロコミック」に連載されていた漫画である。すがやみつるは、その後パソコン入門書などの大人向け学習漫画、娯楽小説作家などを経て、2005 年、54 歳から大学、大学院で学び直して 2013 年から京都精華大学マンガ学部教授に就任し、2020 年にはマンガ学部を離れて同大学国際マンガ研究センター教授に就任した。35 年ぶりの描き下ろし単行本が、ゲームセンターあらしによる Python の入門漫画であった。

(2) Kivy

Python の GUI ライブラリには標準ライブラリの Tkinter 以外にも様々な外部モジュールがあり、近年人気が高いライブラリが Kivy である。パソコンだけではなく、スマホ用 OS の Android と iOS、そして Rasbpberry Pi でも動作するという汎用性の高さがメリットである。ただし、レイアウト用の言語 Kv Language を覚える必要があり、難易度は若干高い。Python でスマホアプリを開発したい場合には、Kivy を使うのが良いであろう。ただし、スマホアプリの開発環境を構築する必要がある。詳しくは、Kivy ドキュメントの翻訳[2]を参照。

【文献】

すがやみつる(2020)『ゲームセンターあらしと学ぶ プログラミング入門 まんが版こんにちは Python』日経 BP

1 http://www.m-sugaya.jp/manga_python/

2 https://pyky.github.io/kivy-doc-ja/

第 10 回　整列アルゴリズム（クイックソート）

バブルソートよりも高速なクイックソートについて学ぶ。

【クイックソート】
クイックソートは、データの比較と交換回数が少なく，非常に効率良くソートできる整列アルゴリズムである。1960 年にアントニー・ホーアが開発した。クイックソートは、とても高速な並べ替えアルゴリズムで、多くのプログラムで利用されている。非常に有名なアルゴリズムなので「クイックソート」で検索すると多くのページがヒットする。先週学習したバブルソートの平均計算量は、データ数 n に対しておよそ n^2 に比例する（このことを $O(n^2)$ と書く）のに対して、クイックソートの平均計算量は $O(n\log n)$ であり、クイックソートの方が圧倒的に速く、データ数が増えるにつれてバブルソートとクイックソートの計算速度の差が開く。今回はクイックソートを学習し、バブルソートとクイックソート以外のソートアルゴリズムについては「発展」で簡単に触れる。

クイックソートの基本的な処理手順は，次の通りである。

(1) 配列の中から基準値（枢軸、ピボットと呼びます）を 1 つ決める。
(2) 枢軸より小さい要素を枢軸より前に，枢軸より大きい要素の枢軸より後ろに移動する。
(3) 枢軸より前方と後方のそれぞれについて、(1) の処理を再帰的に繰り返す。

この処理を Python で実現する。実現方法はいくつかあるが、ここでは，次の手順に従う。

(1) 整列対象配列の中央にある要素を枢軸とする。
(2) 配列を左から順に調べ、枢軸以上の要素を見つけ、その位置を変数 i に格納する。
(3) 配列を右から順に調べ、枢軸以下の要素を見つけ、その位置を変数 j に格納する。
(4) i <= j なら、i 番目の要素と j 番目の要素を入れ替える。i を 1 つ右へ進め、j も 1 つ左を進めた上で (2) に戻る。
(5) i > j となったら，0 番目から i-1 番目と、i 番目から n-1 番目の二つの領域に分け、それぞれに対して再帰的に (1) からの処理を行う。要素数が 1 以下の領域ができたら終了。

【★課題】
クイックソートを実現するプログラムを、
https://paiza.io/projects/pGSKgJEOnM6FH-42805E0g
にアクセスして、????の箇所を編集することで完成させてください。????以外の場所は変えないこと。完成したプログラムを ToyoNet-ACE に提出してください。

```
def quick(left, right):
    pivot = list[(left + right) // 2] # 枢軸
    print(f"領域({left},{right}):{list[left:right+1]}をクイックソート、枢軸は{pivot}")
    i = left # 配列を左から走査する変数
    j = right # 配列を右から走査する変数
    while i <= j:
        while list[i] < pivot: # 枢軸以上の値が見つかるまで i を右へ進める
            i += 1
        while ???? > ????: # 枢軸以下の値が見つかるまで j を左へ進める
            j -= 1
```

```
        if i <= j:
            # 要素の交換
            print("{0} の{1}と{2}を交換".format(list,list[i],list[j]))
            list[i], list[j] = list[j], list[i]
            i += 1
            j -= 1
    # left から j 番目のグループを再帰的にクイックソートする
    if left < j:
        quick(????, ????)
    # j+1 から right 番目のグループを再帰的にクイックソートする
    if left < j+1 < right:
        quick(????, ????)

list = list(map(int,input().split()))
print("{0}のクイックソート".format(list))
quick(0, len(list)-1)
print("ソート完了: {0}".format(list))
```

入力：8 4 3 7 6 5 2 1
出力：[8, 4, 3, 7, 6, 5, 2, 1]のクイックソート
領域(0,7):[8, 4, 3, 7, 6, 5, 2, 1]をクイックソート、枢軸は 7
[8, 4, 3, 7, 6, 5, 2, 1] の 8 と 1 を交換
[1, 4, 3, 7, 6, 5, 2, 8] の 7 と 2 を交換
領域(0,5):[1, 4, 3, 2, 6, 5]をクイックソート、枢軸は 3
[1, 4, 3, 2, 6, 5, 7, 8] の 4 と 2 を交換
[1, 2, 3, 4, 6, 5, 7, 8] の 3 と 3 を交換
領域(0,1):[1, 2]をクイックソート、枢軸は 1
[1, 2, 3, 4, 6, 5, 7, 8] の 1 と 1 を交換
領域(2,5):[3, 4, 6, 5]をクイックソート、枢軸は 4
[1, 2, 3, 4, 6, 5, 7, 8] の 4 と 4 を交換
領域(3,5):[4, 6, 5]をクイックソート、枢軸は 6
[1, 2, 3, 4, 6, 5, 7, 8] の 6 と 5 を交換
領域(3,4):[4, 5]をクイックソート、枢軸は 4
[1, 2, 3, 4, 5, 6, 7, 8] の 4 と 4 を交換
領域(6,7):[7, 8]をクイックソート、枢軸は 7
[1, 2, 3, 4, 5, 6, 7, 8] の 7 と 7 を交換
ソート完了: [1, 2, 3, 4, 5, 6, 7, 8]

入力：21 6 15 2 29 31 89 65 43 70 54 12 28 13
出力：[21, 6, 15, 2, 29, 31, 89, 65, 43, 70, 54, 12, 28, 13]のクイックソート
（中略）
ソート完了: [2, 6, 12, 13, 15, 21, 28, 29, 31, 43, 54, 65, 70, 89]

最初に、0 番目から 7 番目の全体に対してクイックソートを実施する。枢軸は，(0 番目＋7 番目) // 2 で 3 番目、つまり 7 を選択する。変数 i により配列を左から調べ、枢軸より大きい 8 が 0 番目にあるのを見つける。また、変数 j により配列を右から調べ、枢軸より小さい 1 が 7 番目にあるのを見つける。これらを交換する。

1, 4, 3, 7, 6, 5, 2, 8

続いて、変数 i は 1 番目から右へ、変数 j は 6 番目から左へと調べる。変数 i により 3 番目の 7 を発見、変数 j により 6 番目の 2 を発見、これらを交換する。

1, 4, 3, 2, 6, 5, 7, 8

続いて、変数 i は 4 番目から右へ、変数 j は 5 番目から左へと調べる。変数 i は 6 番目に 7 を発見、変数 j は 5 番目に 5 を発見する。しかし、この時点で変数 i と変数 j は左右が入れ替わってしまったので、交換はしない。ここで、変数 j の値が 5 なので、0 番目から 5 番目と 6 番目から 7 番目に領域を分割する。

1, 4, 3, 2, 6, 5 | 7, 8

分割された領域について、まずは左側である 0 番目から 5 番目に対して再帰的にクイックソートする。枢軸は 3 を選択する。変数 i は 0 番目から右に調べ，枢軸より大きい 4 を 1 番目に見つけする。変数 j は、5 番目から左に調べ、枢軸より小さい 2 を 3 番目に見つける。これらを交換する。

1. 2, 3. 4. 6, 5 | 7. 8

変数 i は右隣の 2 番目へ、変数 j も左隣の 2 番目に来る。ここで、プログラムの上では、2 番目と 2 番目を交換するという処理が発生している。

1, 2, 3, 4. 6. 5 | 7, 8

さらに i は 3 番目へ、j は 1 番目へと進む。この時点で i と j の左右が逆転したので、さらに領域を分割する。j の値が 1 なので 0 番目から 1 番目の[1, 2]と 2 番目から 5 番目の[3,4,6,5]に分割する。

1, 2 | 3, 4, 6, 5 | 7, 8

[1,2] の領域については、1 と 1 を交換して終了する。以降同様に、再帰的な処理を繰り返す。[3,4,6,5]の領域については、さらに領域が分割されて[4,6,5]の領域で 6 と 5 が交換される。その結果、最終的にはきちんと並び替えられてソートが完了する。

【無限ループ】

今回のプログラムには while のループ（繰り返し）があるため、プログラムが正しく書けていないと「無限ループ」が発生する可能性がある。正常なプログラムは、並べ替えが完了した時点でプログラムが終了するようにループができているが、どこかで間違えているために、ループが終了しなくなってしまうことを「無限ループ」と言う。たとえば、

```
while 3 > 2:
    処理
```

というようなループがあるとする。この最初の while 文の条件式「3 > 2」は常に True なので、中の「処理」を何度繰り返してもこのループを抜けることはなく、「処理」の中にプログラムを終了する条件が書かれていない限りは、このプログラムはいつまでも終了しない。これが「無限ループ」である。

Paiza では、一定時間内にプログラムが終了しないと強制的に終了し、「Timeout」と表示される。「Timeout」が出た場合には、無限ループである可能性が高い。

【発展：クイックソートよりも優れたソートアルゴリズム】

クイックソートは平均的には高速にソートできるが（平均計算量 O(nlogn)）、運が悪ければ（ピボットがうまく選ばれなければ）、あまり高速ではなくなる（最悪計算量 $O(n^2)$）。平均計算量、最悪計算量ともに O(nlogn)と性能の良いソートアルゴリズムには、マージソート、ヒープソート、イントロソートなどがある。このように、近年はソートの性能を評価する指標として平均計算量だけでなく、最悪計算量、メモリ使用量、安定性など、様々な指標を総合的に評価して選ぶようになり、以前は「速いソートアルゴリズムといえばクイックソート」というほどクイックソートは有名なアルゴリズムであったが、クイックソート以外のソートアルゴリズムが採択されることが多くなっている。

Python では、リスト型のメソッド sort()と組み込み関数 sorted()が用意されている。そのアルゴリズムには、マージソートを元に Tim Peters が 2002 年に Python のために開発したティムソート(Timsort)が採用されている。これは、最悪・平均計算量ともに O(nlogn)であり、さらに現実世界ではよくあるような、ある程度整列されているデータに対する実行速度がマージソートよりも速いとされている。また、要素数が少ないときには挿入ソートを使う。挿入ソートは $O(n^2)$であるが、要素数が少ない場合には高速であるためである。Python のソースコードリポジトリの中にティムソートのドキュメントがある[1]。ティムソートは、Java や Swift にも採用されている。

【発展：Python の実装】

Python はプログラミング言語の名称であり、Python で書かれたプログラムを動かすためのソフトウェア（実装）は一通りではない。その中のいくつかを紹介する。

(1) CPython[2]
Python の標準的な実装（リファレンス実装）であり、C 言語で実装されている。Python と言えばこの実装を意味し、他の実装を区別するときには CPython と表記される。バイトコードインタプリタであり、バイトコードが仮想マシンで実行されるが、バイトコードへの変換は暗黙に行われるために、ユーザーは通常コンパイラを意識しない。

(2) PyPy[3]
JIT（Just-in-Time）コンパイラ（実行時コンパイラ）によって、実行時にコードのまとまりを機械語に逐一コンパイルすることで高速に動作する。PyPy のサイトでは、グイド・ヴァン・ロッサムによる"If you want your code to run faster, you should probably just use PyPy."という言葉が引用されている。

(3) Jython[4]
Java で実装され、Java クラスライブラリを利用できる。Python のプログラムを Java バイトコードにコンパイルできる。

(4) IronPython[5]
C#で実装され、.NET Framework および.NET Framework 互換の Mono 上で動作する。.NET Framework のクラスライブラリを利用できる。

(5) Cython[6]
Python を拡張したプログラミング言語であり、Python のソースファイルを C のコードに変換し、コンパイルすることで Python の拡張モジュールとして出力する。Python のコードの一部に静的な型を宣言することで、コンパイル後のコードの実行速度を高速化できることがある。

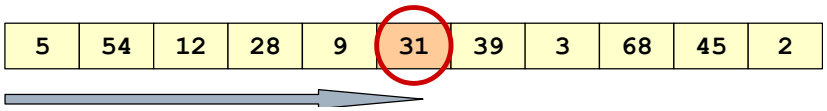
---

1 https://github.com/python/cpython/blob/master/Objects/listsort.txt
2 https://www.python.org/
3 https://www.pypy.org/
4 https://www.jython.org/
5 https://ironpython.net/
6 https://cython.org/

第 11 回　探索アルゴリズム

探索アルゴリズムについて学ぶ。

【線形探索（Linear Search)】
配列の中からある特定の値を持った要素を探すアルゴリズムを、一般に探索アルゴリズムと呼ぶ。探索アルゴリズムには様々なものがあり、もっとも単純な方法が「線形探索」である。線形探索は、配列の先頭から順番に要素を調べていき、探索している数値があるかどうかを調べる。
以下に、線形探索のイメージ図を示す。配列の先頭から順番に調べていき、探索したい値（この例では 31）を探索する。



★線形探索を実現するプログラムを実行しましょう。これまでの知識を活用して「ここではこういう処理をしているのだな」と考えながら打ち込むと良いです。

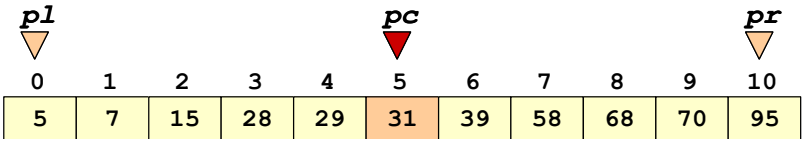
```
import sys
list = list(map(int,input().split()))
key = int(input())
for i in range(len(list)):
    if list[i] == key:
        print(f"探索成功：配列の {i+1} 番目です。")
        sys.exit()
print("探索に失敗しました。")
```

入力：5 54 12 28 9 31 39 3 68 45 2
　　28
出力：探索成功：配列の 4 番目です。

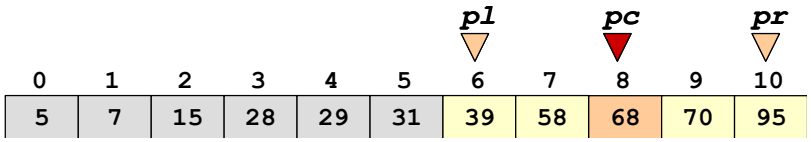
入力データには、1 行目に探索をするデータの列をスペースで区切って入れて、2 行目には探索する数を入れる。
sys.exit() は、プログラムを終了する命令である。sys モジュールの exit 関数を使うので、最初に import sys で sys モジュールを読み込んでいる。このプログラムは、2 つ以上の同じ値がある場合にも、最初の 1 つを見つけた時点で終了する。

【二分探索（Binary Search)】
線形探索では，配列の要素の並び順には何も制約がなかった。しかし、これをあらかじめ昇順に整列させておくことで，「二分探索」と呼ばれる、線形探索よりも効率の良い方法で探索できる。たとえば、10 億個のデータから探索する場合には、線形探索では平均で 5 億回、最悪で 10 億回の探索が必要であるが、二分探索であれば、2 の 30 乗が約 10 億なので、10 億個のデータから 30 回以内の探索で目的のデータが見つかる。5 億回と 30 回でどっちが速いかは考えるまでもない。このように、データの数が増えると探索効率の差が歴然とする。
二分探索とは、あらかじめ整列されたデータに対して、探索対象の中央に位置するデータと探索したいデータの大小を比較して探索範囲を半分に絞るという手順を繰り返すことにより、探索したいデータを探すというアルゴリズムである。

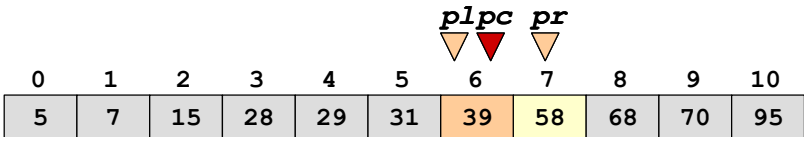
二分探索のイメージ図を以下に示す。二分探索を実行するために、探索範囲の左端を表す変数 pl、探索範囲の右端を表す変数 pr、探索範囲の中央を表す変数 pc を用いる。以下の配列から、39 を探索する場合を考える。まず、pl を左端の 0 番目に、pr を右端の 10 番目に、pc を中央の 5 番目にセットする。



ここで、pc が指し示す 31 と、探索対象の 39 の大小を比較する。探索対象の 39 は 31 より大きいので、31 の右隣から右端までの範囲を新たな探索対象とする。そのために、pl を pc の 1 つ右隣へ移動させ、pc を新 pl と pr の中間にセットする。



ここで、pc が指し示す 68 と、探索対象の 39 の大小を比較する。探索対象の 39 は 68 より小さいので、pl から，68 の 1 つ左隣までを新たな探索対象にする。pc は、pl と pr を足して 2 で割った値を切り捨てて、pl と同じ 6 番目とする。



ここで、pc が指し示す 39 は探索対象に等しいので、探索を終了する。

さて、探索に失敗した場合、何を終了の条件とすればよいだろうか。例えば、上記の例で 40 を探す場合を考えてみよう。pl と pr の位置関係がどうなったときに探索を終了させればよいでしょうか？

【★課題】

二分探索を実現するプログラムの????部分に入る適切なプログラムコードを記述して完成させ、実行して下さい。入力は、小さい数から大きい数へと順番に並んでいる必要があることに注意すること。完成したら、ToyoNet-ACE に提出して下さい。

```
import sys
list = list(map(int,input().split()))
key = int(input())
pl = 0
pr = len(list) - 1
while pl <= pr:
    pc = (pl + pr) // 2
    if ????:
        print(f"探索成功：配列の {pc+1} 番目です。")
        sys.exit()
    elif ????:
        pl = ????
```

```
    else:
        pr = ????
print("探索に失敗しました。")
```

入力：5 7 15 28 29 31 39 58 68 70 95
　　39
出力：探索成功：配列の 7 番目です。

【考え方】
　まずは、二分探索の図と説明をよく見直して、入力例に対してどのように pl，pc，pr などのパラメータが変化するのかをよく考えてみましょう。頭の中で考えるだけでなく、紙に書きながら整理して考えると良い。いきなり pl とか pc のような変数で考えるのが難しい場合には、まずは pl や pc に具体的な数を入れて考えて、それからどの変数が当てはまるのかを考えると良い。そして、????に入る式を考えて、プログラムコードに記入して実行し、うまく動くかどうかを確認して、うまく動かなければ、なぜうまく動かないのか、実際に自分のプログラムに入力する値を入れて手で計算しながら動かしてみると良い。この程度の長さのプログラムであれば、そのように「紙と鉛筆を使って実際にプログラムの動きをなぞってみる」という検証は有効であり、そのような経験を通してプログラミングの考え方を身につけることになる。
　「インデックス」と「リストの要素」を混同する答案が多い（インデックスを書くべきところに要素を書く、あるいはその逆）ので、注意すること。たとえば、list[i]は、list という名前のリストの「インデックス」が i の位置に入っている「リストの要素」を意味することになる。たとえば、

list = [5 7 15 28 29 31 39 58 68 70 95]

のときに、

list[0] = 5
list[1] = 7
list[2] = 15

となる。わからない人は、「コンテナのデータ型」の授業をよく復習すること。

【発展：ハッシュ探索と辞書】
　二分探索よりもさらに効率の良い探索方法に「ハッシュ探索」がある。探索時間を比較すると、線形探索はデータの長さに比例し（O(n)）、二分探索はデータの長さの対数に比例し（O(logn)）、ハッシュ探索はデータの長さに関わらず一定である（O(1)）。ハッシュ探索では、データからハッシュ関数（任意のデータから、別の値を得る関数）によって得られる「ハッシュ」（ハッシュ値）を計算して、そのハッシュからハッシュ表（ハッシュテーブル）によって定まるメモリの場所にデータを格納する、というアルゴリズムである。最初から決められた場所にデータを格納するため、探索する時間はその場所を計算する時間だけとなり、データの長さに依存しない。
　検索エンジンで検索をすると、一瞬で検索した文字列が含まれるウェブサイトの一覧が得られるのは、検索エンジンのクローラがウェブサイトを取得してキャッシュとして保存し、インデックスを作るからである。このときにハッシュ表を使うことで、膨大なキャッシュから目標とする検索文字列が含まれるインデックスを速やかに探し当てることができる。

　Python では、辞書型でハッシュ表によってデータが管理されている。コンテナのデータ型については、リストを中心に扱ってきたが、辞書型について簡単に説明する。辞書型のデータは、たとえば

dict = {'key1': 12, 'key2': 24, 'key3' : 135}

のように「キー」と値によってデータを管理する。そして、dict[‘key1’]のようにキーをインデックスとして値を得る。リストやタプルのようなシーケンス型では、要素の「何番目か」という数をインデックスとして値を得る。それに対して、このようにキー（数とは限らない）をインデックスとして値を得るデータ構造を、一般に「連想配列(associative array)」「辞書」「マップ」などと呼ぶ。すなわち「キーと値の組」のデータ構造である。

Python では、辞書型でこの連想配列が実現されていて、ハッシュ探索を使ってキーから速やかに値を得ることができる。その仕組みは、次のようになっている。キーはハッシュ関数によってハッシュ表の大きさにあわせた整数のハッシュに変換され（ここで使われるハッシュ関数は、暗号で使われるような複雑なものではない）、そのハッシュが定めるメモリの場所にデータが格納される。ハッシュ表の大きさは 2 の n 乗となっている。たとえば、ハッシュ表の大きさが 128 の場合は、0 から 127 までの整数のハッシュが得られ、キーから計算されるハッシュが 58 であれば、58 番目のデータに直接アクセスする。ここで、異なるデータで同じハッシュが計算される「ハッシュの衝突」が生じた場合には、オープンアドレス法というアルゴリズムによって「次に開いている場所」が探索されて、そこに格納される。ハッシュ表のあいている場所が少なくなると、ハッシュ表が 2 倍に拡大され、再構築(rehash)される。詳しいハッシュの計算方法については、Python ソースコードの辞書オブジェクト（dictobject.c）[1]にコメントで解説されている。

このように、Python の辞書型はハッシュ表によってデータが管理されているため、大量のデータを取り扱う場合でもキーから効率的にデータを探索、読み出し、書き込み、追加、削除することができる。また、Python の辞書型でキーとして使えるのは「ハッシュ可能なオブジェクト」に限られる。数値や文字列はハッシュ可能であるが、リストはハッシュ可能でないのでキーとして使うことができない。変更可能なオブジェクト（mutable object）はハッシュ不可能である。タプルは変更不可能でハッシュ可能なので、リストを「キー」として使いたい場合には、タプルに変換するのが良いであろう。

[1] https://github.com/python/cpython/blob/master/Objects/dictobject.c

第 12 回　オブジェクト指向プログラミング

オブジェクト指向プログラミングの考え方について学ぶ。

【オブジェクト指向プログラミング】
Python はオブジェクト指向プログラミングができる。オブジェクト指向プログラミングとは、データとメソッドをひとつにまとめてオブジェクトとして、オブジェクトを中心にプログラミングをすることである。オブジェクト指向プログラミングに特有の用語について、簡単に解説する。
オブジェクトの種類を「クラス」と呼び、あるクラスに基づいて生成された具体的なオブジェクト（物）を「インスタンス」と呼ぶ。たとえば、「ビールとはこういうものである」ということが書かれている酒税法は「ビールクラス」を定めていて、実際に店で売られている１つ１つの缶ビールは、「ビールクラス」に基づいて生成された「ビールインスタンス」（ビールオブジェクト）である、と考えることができる。
Python では、組み込みのクラスとしてたとえば「int クラス」「str クラス」のようなものが定められている。さらに、そのクラスが持つ状態（変数）や、そのクラスがどういう動作をするか（メソッド）が定義されている。そして、あるクラスに基づいて生成されたオブジェクト（インスタンス）は、呼び出されたクラスで定義されているメソッドを使って、操作することができる。
たとえば s = “Hello” という文を実行することで、s という変数に「Hello という文字列が代入される」とこれまで説明をしてきたが、これは str クラスを元に Hello という具体的な文字列の「オブジェクト（インスタンス）」が生成された、ということになる。この Hello という文字列は、str クラスを元に生成されたオブジェクトであるため、str クラスで定義されているメソッドを使うことができる。たとえば、str クラスには replace というメソッドが定義されているので、Hello という文字列に replace というメソッドを適用することができる。なお、replace というメソッドは文字列の置換をするメソッドであり、たとえば次のプログラムの出力は

```
s = "Hello"
print(s.replace("H", "h"))
```

出力：hello

となる。

これまでに使ったメソッドには、str クラスに定義されている split というメソッドがある。これは、文字列をスペースやタブなどの「ホワイトスペース」で区切ったリストを生成するものであり、オプションによって区切り文字を指定することもできるが、詳しくは触れない。これまでのプログラムで使われていた

```
list = list(map(int,input().split()))
```

という文は、input().split() のところで、組み込み関数の input() で得られた文字列が str クラスのオブジェクトとして生成されて、そのオブジェクトに対して、split()というメソッドが適用されている。
このように、str クラスを定義して、そのクラスに「メソッド」を定義することで、文字列に対して replace や split のような「共通する操作」を定義することができる、というところがオブジェクト指向プログラミングの特徴である。

Python で変数に代入される値は、基本的にはどのような値も「オブジェクト」であり、つまりなんらかの「クラス」を元に生成された「インスタンス」である。そして、ある変数がどのようなクラスから生成されたインスタンスであるかは、type を使って確認することができる。

★次のプログラムを実行してみましょう。

```
s = "Hello"
print(type(s))
i = 1
print(type(i))
n = 1.5
print(type(n))
list = [1,2,3]
print(type(list))
b = True
print(type(b))
```

【クラスの作成】

Python では、「数」「文字列」「リスト」といった組み込みのクラスだけではなく、クラスを自作することができる。様々な種類のデータをオブジェクトとしてとらえて、そのオブジェクトの性質を「クラス」として抽象化し、そのオブジェクトにはどのような属性があるか、そしてそのオブジェクトに対してはどのような操作ができるか、ということをプログラミングしていくことが、オブジェクト指向プログラミングの基本的な考え方である。

★次のプログラムを実行してみましょう。ファイルは 2 つあります。コードは
https://paiza.io/projects/3a6G1hHwwmP-ms2k6Prsog

point.py

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def up(self):
        self.y += 1
    def down(self):
        self.y -= 1
    def right(self):
        self.x += 1
    def left(self):
        self.x -= 1
    def pos(self):
        return self.x, self.y
```

Main.py

```
import point
a = point.Point(2,3)
```

```
a.up()
a.up()
a.right()
a.down()
print(a.pos())
```

出力：(3, 4)

ターミナルで実行するときには、Main.py と point.py を同じディレクトリに置く。.py の拡張子のファイルはモジュールとして認識されて、Main.py の import point で point.py がモジュールとして読み込まれる。a = point.Point(2,3)では、後で説明されるように point モジュールの Point クラスが読み込まれる。

point.py すなわち point モジュールの Point クラスは、二次元座標平面上の点(x,y)をあらわすクラスである。class Point: で、Point という名称のクラスを定義することを宣言する。この文をヘッダとして、インデントによってクラスの中身が書かれている。そして、クラスの中には def ではじまるメソッドの定義が 6 個ほど書かれている。

最初のメソッドは def __init__(self, x, y): である。__init__ という名称のメソッドは、クラスからオブジェクトが生成されるときに最初に実行される。このメソッドには 3 つの引数があり、その 1 つ目は self である。self は「自分自身」つまり「そのオブジェクト自身」である。クラスのメソッドには必ず最初の引数に self を入れる。そして、(self, x, y) のように、self の次に x と y という引数があり、これはオブジェクトを生成するときにユーザーが明示的に指定をする。Main.py で

```
a = point.Point(2,3)
```

が実行されると、point モジュールの Point クラスから (2,3) を引数としてオブジェクトが生成され、この __init__(self, x, y) が呼び出されることにより、x=2、y=3 として処理が実行される。そして、self.x = x では、その自分自身 (self) の x という属性 (インスタンス変数) が x、つまり 2 であると設定される。この self.x というのは、たった今作成しているオブジェクト、つまり Main.py では a という変数の属性なので、Main.py からは a.x として呼び出すことができる。次に、self.y = y によって、y=2 として得られた引数が、self.y に代入される。このようにして、a という変数の座標平面上の位置を表現している。

次に、def up(self): では、up という名前のメソッドが定義されている。このメソッドは、座標平面上を上に 1 つ進む、つまり y 座標の値を 1 増やす、という操作をするメソッドである。y 座標の値は self.y に格納されているため、その値に 1 を足すことで実現している。同様に、down という名前のメソッドは下に進む、つまり y 座標の値を 1 減らしている。また、left という名前のメソッドは左に進む、つまり x 座標の値を 1 減らし、right という名前のメソッドは右に進む、つまり x 座標の値を 1 増やしている。

```
a.up()
a.up()
a.right()
a.down()
```

の箇所では、先ほど(2,3)の位置に設定された a という変数が、上に進んで、上に進んで、右に進んで、下に進んでいる。そして、最後の

```
print(a.pos())
```

では、a.pos() の値を表示している。ここでは、a は Point クラスのインスタンスなので、def pos(self): で定義されているメソッドが呼ばれて、x 座標と y 座標の組（タプル）が返され、それが表示される。

【★課題】
上のプログラムの Main.py を修正して、

(1) 座標（7,5）に設定して
(2) 左に移動して
(3) 上に移動して
(4) 左に移動して
(5) 上に移動して
(6) 下に移動して
(7) 座標を表示する

という動きをするプログラムに書き換えて、ToyoNet-ACE の「レポート」から Main.py **だけ**を提出してください。point.py は編集も提出もしないこと。

【発展：スタイルガイドの命名規約】
Python のコードを読みやすくするためのスタイルガイドとして PEP8 がある[1]。PEP は Python Enhancement Proposal の略で、Python の機能拡張のためにコミュニティが話し合ってとりまとめた文書がまとめられたもので、その 8 番目である。そこには、たとえばインデントのそろえ方、1 行の長さなどが書かれている。命名規約として書かれているものとして、いくつか抜粋する。

(1) 小文字のエル、大文字のオー、大文字のアイなどを単一の文字で変数名にしない。
(2) ASCII の文字を使う。
(3) モジュールの名前は小文字のみの短い名前にする。
(4) クラスの名前は単語の初めの文字だけを大文字にする CapWords 方式を使う。
(5) 関数や変数の名前は小文字のみにして、必要に応じて単語をアンダースコアで区切る。
　メソッドとインスタンス変数も同様にする。

今回の課題では、モジュールの名前を point クラスの名前を Point としたのは、この命名規約に従った。

スタイル以前の問題として重要なことは、キーワード（予約語）は関数や変数などの名前として使えない、ということである。キーワードとは、たとえば if, True, return のように Python が特別な意味を持つ単語として解釈するものであり、キーワードの一覧を次のコマンドで確認できる。

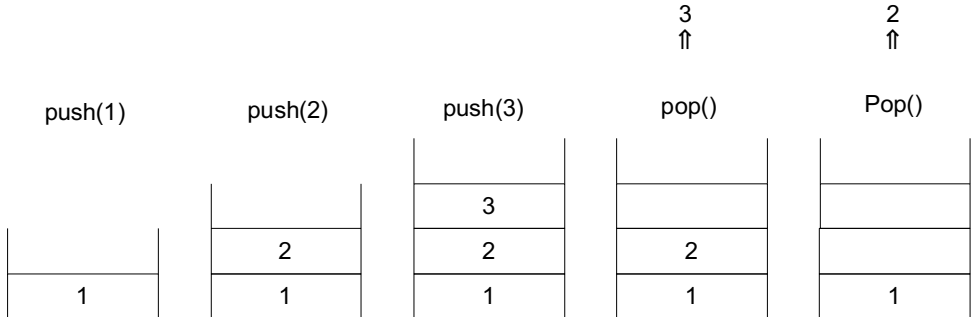
```
import keyword
print(keyword.kwlist)
```

[1] PEP8 の原文 https://www.python.org/dev/peps/pep-0008/ と和訳 https://pep8-ja.readthedocs.io/ja/latest/

第 13 回　スタックとキューのデータ構造

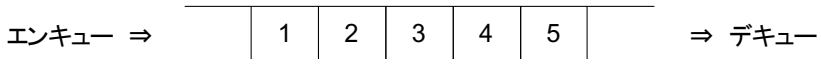
スタックとキューのデータ構造を Python で実現する方法について学ぶ。

【スタック】
スタック(Stack；物を積み上げた山)は、「後入れ先出し（LIFO：Last In, First Out)」型のデータ構造である。つまり、最後に格納したデータが最初に取り出されるデータ構造である。データを格納する操作をプッシュ、データを取り出す操作をポップと言う。スタックの動作イメージを以下に示す。



日常生活の例：食堂の皿の山
プログラムの例：メソッド呼び出し、操作の「元に戻す」「繰り返し」

【キュー】
キュー(Queue；待ち行列)は、「先入れ先出し（FIFO：First In, First Out)」型のデータ構造、すなわち最初に格納したデータから順に取り出されるデータ構造である。データを格納する操作をエンキュー、データを取り出す操作をデキューと言う。キューの動作イメージを以下に示す。



日常生活の例：スーパーのレジ等の待ち行列（先に並んだ客から処理される）
英語では First come, first served (FCFS)
プログラムの例：プリンタの実行待ちジョブのキュー

【両端キュー】
両端キュー（double-ended queue）またはデック（deque）とは、先頭または末尾で要素を追加・削除できるキューである。両端キューを使えば、スタックとキューの操作をいずれも実現できる。Python では、両端キューが collections.deque で実装されている。Python の組み込みコンテナ（list, tuple, dict, set）にはそなわっていないコンテナのデータ型を集めたモジュールに、collections モジュールがある[1]。ドキュメントには次のように書かれている。

> Deque とは、スタックとキューを一般化したものです（この名前は「デック」と発音され、これは「double-ended queue」の省略形です)。Deque はどちらの側

[1] https://docs.python.org/ja/3/library/collections.html

からも append と pop が可能で、スレッドセーフでメモリ効率がよく、どちらの方向からもおよそ O(1) のパフォーマンスで実行できます。
　list オブジェクトでも同様の操作を実現できますが、これは高速な固定長の操作に特化されており、内部のデータ表現形式のサイズと位置を両方変えるような pop(0) や insert(0, v) などの操作ではメモリ移動のために O(n) のコストを必要とします。

たとえば、[1,2,3,4,5]というリストを考える。このリストに 6 というデータを追加すると、[1,2,3,4,5,6]となる。スタックであれば、ここからデータを取り出すときには一番最後に追加した最後のデータ 6 を取り出せば良いが、キューであれば、最初のデータ 1 を取り出すことになる。その場合に、1 を取り出してあいたところに 2 を移動し、さらにあいたところに 3 を移動し、といったデータの移動をするために、時間がかかってしまう。このことが「メモリ移動のために O(n) のコストを必要とします」と表現されている。
Deque のデータ型では、このような問題が解決されて「どちらの方向からもおよそ O(1) のパフォーマンスで実行」できるとされている。つまり、メモリを「詰めていく」という内部処理を必要としないデータ構造となっている。したがって、先頭や途中にデータを追加したり削除したりすることが多い場合（そして大量のデータを取り扱うとき）には、list よりも deque を使う方が良い。
このような計算資源の効率的な使い方は、日常的なプログラミングではほとんど気にする必要がない。なぜならば、コンピュータの処理は高速であり、通常は一瞬で計算が終わるためである。一方、同じような処理を何度も繰り返すような場合や、大量のデータを処理する場合には、処理速度が問題になることがある。たとえば、10 日間かかる計算が 2 秒で終わるのであれば、そのメリットは大きい。アルゴリズムやデータ構造を効率化することで、このようなプログラムの高速化に役立つことがある。
Deque に用意されているメソッドの中から、いくつかピックアップする。

append(x)	x を deque の右側に付け加える。
appendleft(x)	x を deque の左側に付け加える。
index(x)	deque 内の x の位置を返す。
insert(i, x)	x を deque の位置 i に挿入する。
pop()	deque の右側から要素をひとつ削除し、その要素を返す。
popleft()	deque の左側から要素をひとつ削除し、その要素を返す。
remove(value)	value の最初に現れるものを削除する。

★deque を使って次のプログラムを動かしてみよう。

```
import collections
queue = collections.deque(["a", "b", "c"]) # a, b, c という並びのキューを作成
queue.append("d") # d を右側に付け加える
queue.insert(1,"e") # 1番目にeを挿入。「1番目」はどこになるか？
print(queue.pop()) # 右側から削除して、その要素を表示。
print(queue.popleft()) # 左側から削除して、その要素を表示。
print(len(queue)) # キューの長さを表示
queue.remove("b") # b を削除
print(list(queue)) # キューの状態を表示
```

次のように出力される。

```
d
a
3
['e', 'c']
```

【★課題】
　上のプログラムを参考に、次の動作をするプログラムを作成して、ToyoNet-ACE に提出してください。途中の経過がないプログラムは無効です。

(1) a, b, c, d, e という並びのキューを作成
(2) f を右側に付け加える
(3) c の右側に g を付け加える
(4) キューの一番左から削除して、その要素を表示する
(5) キューの状態を表示

出力：

```
a
['b', 'c', 'g', 'd', 'e', 'f']
```

【発展：競技プログラミング】
　これまでの授業で、だいぶプログラミングに慣れてきたでしょうか。自ら、さらに深く勉強しようという人も出てきていることでしょう。そこで「競技プログラミング」について紹介する。競技プログラミングでは、参加者全員に同じ課題が出題され、与えられた課題を正確にかつより早く解決するプログラムを書くことを競う。国内では、AtCoder という競技プログラミング(https://atcoder.jp/)が有名である。コンテストで良い成績を取るとレーティング、ランキングが上がる。
　参考までに、AtCoder Beginner Contest 149 (2019 年 12 月 29)の問題 C (Next Prime)を示す[1]。これまでの授業の知識で十分に解ける問題なので、ぜひ挑戦してみてください。AtCoder のサイトからコンテストで提出された様々なプログラミング言語の様々な回答を見ることができる[2]。

問題文: X 以上の素数のうち、最小のものを求めよ。
注記: 素数とは、2 以上の整数であって、1 と自分自身を除くどの正の整数でも割り切れないようなもののことです。例えば、2,3,5 は素数ですが、4,6 は素数ではありません。
制約: $2 \leq X \leq 10^5$、入力はすべて整数
入力: 入力は以下の形式で標準入力から与えられる。

```
X
```

出力: X 以上の素数のうち、最小のものを出力せよ。

入力例 1: 20
出力例 1: 23
入力例 2: 2
出力例 2: 2
入力例 3: 99992
出力例 3: 100003

[1] https://atcoder.jp/contests/abc149/tasks/abc149_c
[2] 私が作成した回答例は https://paiza.io/projects/YGWJ685Cd2p6s0qEeuvfzg

【発展：ウェブアプリ開発】

ウェブでアプリを実行するための技術には様々なものがある。大きく分けると、サーバー側（サーバーサイド）で動かすプログラムと、ウェブブラウザ（クライアントサイド）で動かすプログラムがある。

Python でサーバーサイドのウェブアプリを開発するときには、Django（ジャンゴ）、Flask（フラスク）などのフレームワークを導入するのが一般的である。Python のフレームワークには、ログイン機能などのウェブアプリで標準的に使われる機能が備わっているため、効率的にアプリを開発することができる。従来は、ウェブアプリケーションを実行するたびに呼び出す CGI がよく使われていたが、CGI は実行するたびにプロセスを起動するため、最近ではサーバ内のプロセスでウェブアプリケーションを実行する方法が主流である。サーバーの開発環境は Docker のコンテナによって構築し、複数のコンテナを Kubernetes によるコンテナオーケストレーションによって運用管理するという手法が標準となっている。私自身は、研究用のソフトウェア SWRC Fit[1]と EC Fit を Python の CGI プログラムとして公開している。

サーバーにデータを保存する必要がなければ、サーバーの環境を選ばないクライアントサイドプログラミングが手軽である。クライアントサイドプログラミングでは JavaScript が使われ、そこに WebAssembly が組み合わされることがある。

JavaScript は言語仕様が ECMAScript として標準化され、ほぼすべてのメジャーなウェブブラウザでサポートされている。サーバーサイドプログラミングでも、フォームに入力するときの入力チェックなど必要に応じて JavaScript が併用されるため、ウェブアプリを開発するのであれば JavaScript は HTML, CSS と並んで必修科目である。JavaScript はウェブブラウザで直接開発できる点も手軽である。たとえば Chrome ではブラウザを右クリックして「検証」を選ぶことで Console が表示され、直接 JavaScript を実行することができる。

WebAssembly は C, Rust 言語などの様々なプログラミング言語がコンパイルしてウェブブラウザの仮想マシンで実行できるコードを生成するように設計されたもので、主要ブラウザでサポートされている[2]。WebAssembly は JavaScript の代替というよりは、JavaScript と組み合わせて使われるものである。Python の WebAssembly への移植である Pyodide[3]を使うことによって、JavaScript プログラムの中で Python を実行することができる[4]。なお、Node.js によって JavaScript をサーバーサイドで実行することも可能である。

私が JavaScript で作成したパズルを 2 つ紹介する。

(1) 15 パズル[5]

ソースコード[6]を GitHub で読むことができる。

(2) ナンプレ[7][8]

Python でナンプレを解き、問題を作成するプログラムを作成し、そのプログラムで自動作成した問題を難易度別問題集として JavaScript プログラムで出題している。ヒント表示機能では、JavaScript から Pyodide によって Python のプログラムを呼び出している。なお、スマホアプリは Flutter により開発した。

---

1 https://seki.webmasters.gr.jp/swrc/?lang=ja

2 https://webassembly.org/roadmap/

3 https://pyodide.org/en/stable/

4 https://sekika.github.io/2022/08/18/Pyodide/

5 https://sekika.github.io/2020/01/17/15Puzzle/

6 https://github.com/sekika/sekika.github.io/blob/master/js/15.js

7 https://sekika.github.io/kaidoku/ja/sudoku

8 ナンプレは世界的には sudoku として有名であるが、「数独」という名称はニコリが商標登録しているため、日本国内ではニコリが関与しないものはナンプレと表記される。

第 14 回　数値計算アルゴリズム（モンテカルロ法）

今回と次回は、数式などの近似解をコンピュータで求める手法を学ぶ。

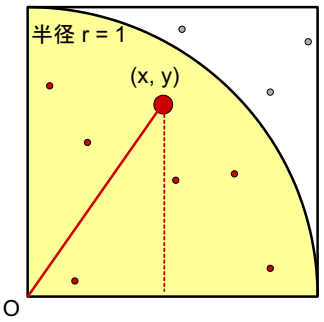
【モンテカルロ法】

まずは、シミュレーションにより近似解を求める「モンテカルロ法」について学ぶ。モンテカルロ法は、多くの回数シミュレーションを実行することで、解析的に解くことが困難な問題であっても近似的な解を求める手法である。

モンテカルロ法は物理学、統計学、工学、生物学などの分野で使われる。金融の分野では、コーポレート・ファイナンスや数理ファイナンスの分野で使われる[1]。

ここでは、円周率πをモンテカルロ法により次の手順で求めるプログラムを考える。

(1) 辺の長さが 1 の正方形を考え、この正方形の内側に n 個の点をランダムに打つ。
(2) この正方形の左下の点を中心とする半径 1 の円の 1/4 部分を考え、この 1/4 円の中に入った点の数 count を数える。
(3) count を n で割った値は、1/4 円の面積π/4 の近似値と考えられる。この関係を利用してπを近似的に求める。



【★課題】

モンテカルロ法によりπを求めるプログラムを????の箇所を修正させることによって完成させて、ToyoNet-ACE から提出してください。モンテカルロシミュレーションでπの近似をするという趣旨のプログラムであるから、pi = のところに直接πの近似値を書くような方法ではこの課題を解いたことにはならず、不正解となる。

```
import random
n = int(input())
count = 0
for i in range(n):
    x = random.random()
    y = random.random()
    dist = ????  # (x,y)の原点からの距離の2乗
    if dist <= 1:
        count += 1
pi = ????
print(pi)
```

入力：10
出力：3.2（乱数が使われているので、実行するたびに値が変わる）

変数 dist は原点からの距離ではなくて、その 2 乗としているが、dist <= 1 で「距離の 2 乗が 1 以下」つまり「距離が 1 以下」となる。Random モジュールの random.random 関数は、53 ビット精度の浮動小数点で 0 以上 1 未満の一様乱数を返す関数である[2]。

[1] https://en.wikipedia.org/wiki/Monte_Carlo_methods_in_finance
[2] https://docs.python.org/ja/3/library/random.html

π = 3.1415926535897932… である。入力する値を大きくすると、より精度の高い近似が得られる。入力する値を 10, 100, 1000, 10000, …と増やしていき、どの程度精度が高くなるかを確認してみよう。

【発展：ガウス・ルジャンドルのアルゴリズム】

ガウス・ルジャンドルのアルゴリズムは、円周率の近似値を計算するためのとても収束が速いアルゴリズムで、高橋大介は 2009 年にこのアルゴリズムで円周率を 2 兆 5769 億 8037 万桁計算した[1]。

$$a_0 = 1, b_0 = \frac{1}{\sqrt{2}}, t_0 = \frac{1}{4}, p_0 = 1$$

として、次の反復式を a, b が希望する精度になるまで繰り返す。これは、a と b の算術幾何平均（算術平均と幾何平均を繰り返して作られる数列の極限）を計算していることになる。

$$a_{n+1} = \frac{a_n + b_n}{2}$$
$$b_{n+1} = \sqrt{a_n b_n}$$
$$t_{n+1} = t_n - p_n(a_n - a_{n+1})^2$$
$$p_{n+1} = 2p_n$$

円周率πは、a, b, t を用いて以下のように近似される。

$$\pi \approx \frac{(a+b)^2}{4t}$$

★円周率の近似値を計算するプログラムを実行してみよう。
https://paiza.io/projects/4zOZhkhFjWI5up7Gbe3MaA

このプログラムは、Wikipedia の「ガウス＝ルジャンドルのアルゴリズム」に掲載されていた Python のプログラムを元に、Paiza の入力欄に桁数を入れると小数点以下その桁数まで計算ができるようにしたものである。Decimal モジュールを使うことで、float 型では計算できないような精度の高い計算を可能としている。

このプログラムでは、Paiza 上で円周率を 1 万桁計算できるが、10 万桁計算すると計算に時間がかかりすぎるため Timeout が起きる。手元の PC で実行したところ、10 万桁は 11 秒、100 万桁は 3 分で計算できた。Rust による自作プログラム[2]では、100 万桁を 2 秒以内で計算できた。ただし、16GB メモリの PC では 3 億 2000 万桁の計算が限界であった。

---

1 https://www.hpcs.cs.tsukuba.ac.jp/~daisuke/pi-j.html

2 https://sekika.github.io/2024/03/17/Rust-pi/

第 15 回　数値計算アルゴリズム（ニュートン法）

【ニュートン法】

ニュートン法とは、非線形方程式 f(x)=0 の解を反復法の数値計算によって求める手法であり、収束の早いアルゴリズムとして有名である。適当な初期値 x_0 から始めて、漸化式

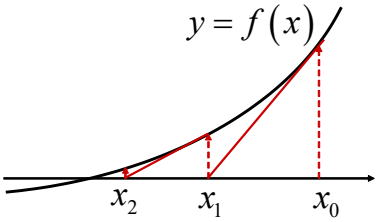
$$x_{k+1} = x_k - f(x_k) / f'(x_k)$$

の収束先から非線形方程式の解を求める。すなわち、y = f(x) のグラフが x 軸と交差するときの x の値を近似的に求めるアルゴリズムである。まず、y = f(x) 上の点 (x_0、$f(x_0)$) を考える。この点における y = f(x) の接線の方程式は、

$$y - f(x_0) = f'(x_0)(x - x_0)$$

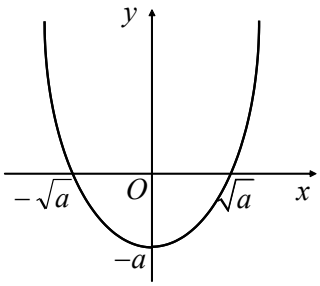
で与えられる（高校数学を思い出して下さい）。

この接線が x 軸と交わる点の x 座標を求めてみましょう。はい、やってみて下さい。



こうして得られた x 座標の値を x_1 と置く。そして同じように、y = f(x) 上の点 (x_1、$f(x_1)$) における接線の方程式を考え、この接線が x 軸と交わる点の x 座標を求め、これを x_2 と置く。こうして順次繰り返していき、$|x_k - x_{k-1}|$ が十分に小さくなった時点で計算を終了する。そのときの x_k の値を近似解とする。

それでは実際に、ニュートン法を用いて整数 a の平方根 $\sqrt{a}$ を求めるプログラムを考える。$\sqrt{a}$ を求めるためには、$y = x^2 - a$ が x 軸と交わる点を求めれば良い。したがって、$f(x) = x^2 - a$ を上記の漸化式に代入し、x_k を求める式を作る。あとはニュートン法に従い、$|x_k - x_{k-1}|$ が十分に小さくなった時点で得られた x_k の値を、a の平方根の近似解とする。



【★課題】

ニュートン法により平方根を求めるプログラムを????に式を入れることによって完成させて、ToyoNet-ACE に提出して下さい。

```
a, x = map(int,input().split())
error = 1e-15
prev_x = x + 1
while (abs(x - prev_x) > error):
    prev_x = x
    x = ????
    print(x)
```

入力：2 10
出力：（最後の行が）1.414213562373095

平方根を求めたい数値は、a に代入される。平方根の近似解を求めるために用いる初期値は x に代入される。前ページの説明では、x_0 に対応する。ニュートン法の漸化式において、x_k と x_{k+1} をともに x とする式を使って、x の値を更新する。つまり、while ループの 1 回目では x_0 から x_1 が計算され、2 回目では x_1 から x_2 が計算される。この計算をする前に、prev_x=x によって x の値を prev_x に保存しておくことで、x - prev_x によって誤差を計算できるようにする。誤差の絶対値 abs(x - prev_x)が許容誤差 error よりも小さくなったところで計算が終了するようにループをまわす。ここで、prev_x の初期値は x+1 としておくことで、必ず最初の 1 回はループを通るようにしている。上の式で a と x_0 に当たる数を入力に指定してプログラムを実行する。

√2 の値を計算してみましょう。正確に計算されているかどうかは import math; print (math.sqrt(2)) とすれば確認できる。なお、この課題はニュートン法による平方根の計算を試みるというものなので、math.sqrt 関数など、他の方法を使った場合は不正解となる。動作確認のためには、√4 のように整数となる計算を試すのも良い。

√2 の計算ができても、√3 の計算ができなければ不正解である。実際に、そのような解答を提出する学生がいるので、√2 以外の値でも、確かめてみること。

ニュートン法についてのいくつかの注釈

(1) 複数の解がある場合には、初期値の選び方によってどの解に収束するのかが変わる。このプログラムの場合は、初期値が正の値なのか負の値なのかによって収束先が決まる。
(2) 解がある場合でも、初期値によっては必ず収束するとは限らない。
(3) n 次元ニュートン法によって n 変数非線形連立方程式を解く手法がある。

【おわりに】

これで授業は終了です。この課題の採点が終わると、課題の平均点が計算され、ガイダンスで示したように課題の平均点から成績が評価されます。課題の平均点と成績評価については、ToyoNet-ACE の成績でこの課題の採点とほぼ同時に見ることができるようになります。

この授業では、ほぼ完成しているプログラムの穴埋めをするという課題をしてきました。プログラミングの考え方について要領がつかめたでしょうか。

プログラミングを習得するためには、半完成プログラムの穴埋めだけではなく、自らプログラムを作ってみる、という経験が重要です。興味を持った学生は、プログラミングの本やネットの記事を参考に、自分で何か作ってみましょう。この授業が足がかりとなり、勉強しやすくなっているはずです。

ちょっとした計算をするときや、定型作業をするときに、この作業は無駄なのでもっと効率化できないだろうか、と考えることはないでしょうか。そのときに、必ずしもプログラミングをしなくても、既存のサービスを使う、作業の流れを工夫する、といったことによって、効率化できる可能性があります。この授業でのプログラミング経験が、そういった作業のアルゴリズムを考える訓練につながっていれば幸いです。