補足資料

2024．05.24 版

序章

● 識別子の命名法のルールと慣行

識別子（identifier，変数や関数，クラスなどに付けられた名前）の命名については，以下のルールと慣行を守って，可読性の高いコードを書くようにします。

(1) 文法のルール（rules，規則）に従います。これは必ず守らなければなりません。

(2) 言語に固有の慣行（conventions）に従います。Python には **PEP8**（Style Guide for Python Code）や Google Python Style Guide というガイドラインがあります。

(3) ライブラリに固有の慣行に従います。たとえば，外部ライブラリに別名を用いる場合には，Numpy では `import numpy as np` のように，別名に `np` を用いることが慣行になっています。

(4) 研究室や会社での慣行に従います。

(5) 個人でも 1 つのプロジェクトやレポートでのプログラムでは一貫した命名法とします。

具体的には，識別子を以下のように命名します。変数の命名法は，他のデータ型名や関数名などにも共通することが多いことに留意してください。

① 半角の英数字と記号_を組み合わせます。
使える文字は，英字アルファベット（`a`～`z`，`A`～`Z`），数字（`0`～`9`），記号は `_` アンダースコア（アンダーバー）のみです。先頭の文字に使えるのは，英字か，アンダースコアに限られます。日本語（漢字やひらがなの全角文字）も変数名として使えますが，使うことはお勧めしません。

② 英文字の扱い
- 小文字と大文字はいつも厳格に区別され，異なる変数名となります。
- 小文字を使うことを基本とします。小文字のほうが読みやすいからです。
- 大文字は語頭あるいは語の繋ぎに使います。慣用的に用いられている場合や，意味を強調したい場合に使うこともあります。定数名にはすべて大文字を使う慣行があります。
- 単一文字の変数名で，小文字の `l`（エル），大文字の `I`（アイ），大文字の `O`（オー）は使わないようにします。なぜか，理由を推察してください。

③ アンダースコア（`underscore`）の扱い
- 語頭に一つあるいは二つのアンダースコアを使うことで意味をもたせることがあります。これは変数やメソッドを `private` とする特別の場合に限定されます。
- 特別なグローバル変数やメソッドでは，前後に `2` つのアンダースコアを付ける名前が用いられています。たとえば，`__name__`。 これらの名前は予約語扱いとします。
- 長い変数名で語の繋ぎに使います。
- 予約語や組み込み関数名などとの衝突を避けるために，変数名の末尾にアンダースコアを一つ付けることがあります。例： `list_`

④ 数字の扱い：数字（`digit, figure`）は文字に含まれます。数字と数（`number`），数値（`value`）をしっかり区別してください。変数名と数を区別する必要から，先頭が数字の名前は変数名に使えません。

⑤ **キーワード（予約語）**と同じ名前は使えません。
キーワード（keywords）は，構文を理解するための鍵となる重要な意味が割り当てられている単語で，使用がシステムで予約されている予約語です。ですから，キーワードは構文で使う以外に，変数名や関数名

などとして使うことのできない識別子です。Python のキーワードは，次の 33 語です。

False True None and as assert break class continue def del elif else except finally for from global if import in is lambda nonlocal not or pass raise return try while with yield

⑥ 広く慣用的に用いられている語がありますので，それらは慣用に従うようにしてください。たとえば，クラスではメソッドの第 1 引数に `self`，あるいは `cls` という語が用いられます。

⑦ 何を表しているか，名前から直ぐに意味がわかるようにします。

⑧ 変数名は，関数名やクラス名などと，一瞥しただけで区別できるようにしておくとよい。変数名は，間違いを防ぐため，関数名やメソッド名と同じ名前としないようにします。

⑨ 語と語のつなぎ方には次の 3 通りがあります。

・snake_case 記法：すべて小文字で，1 個のアンダースコアで語をつなぎます。

・camelCase（lower camel Case）記法：先頭の語の頭文字は小文字，それ以降の単語の先頭の頭文字を大文字とします。

・Pascal（upper Camel Case）記法：すべての語の頭文字を大文字にします。

・名前を空白やドット（.），ハイフン（-）でつなぐことはできません。

⑩ 変数名や関数名は snake_case 記法，クラス名は Pascal 記法で書くという慣行があります。

⑪ メタ構文変数，`foo`，`hoge` などには意味はとくになく，便宜的に使われています。

【Prg】 PEP8：Style Guide for Python Code

PEP8 は，よりよいコーディングのための提案で，スタイルガイド，作法という位置づけです。文法的な規約ではありませんので，必ず従わなければならないということではありません。たとえば，インデント（indentation，字下げ）は文法的なルールですが，半角の空白の個数は任意です。PEP8 では空白 4 文字を推奨していますが，Colab ノートブックでは空白 2 文字が使われています。

クラス名は upper Camel Case で書くことが推奨されていますが，小文字から始めても間違いではありません。たとえば，標準モジュール `datetime` には 同名の `datetime` クラスがあります。ただし，混乱を避け，可読性を高めるために，できるだけ推奨に従うことをお勧めします。

● 論理的エラーの修正，デバッグの方法

① アルゴリズムの理解とプログラムの流れの記述に間違いがないかを確認します。

② プログラムの中でバグが起きている可能性の高い，怪しいブロックを特定します。バグが発生しやすいブロックには次のような箇所があります。

・アルゴリズムをきちんと理解しないままコードを書いたブロック

・繰り返しや条件分岐のブロック。特に入れ子のブロックで，繰り返しの条件や条件分岐の判断を間違えていることが多い。

・関数やクラスの定義。引き数の数と順序

③ 怪しいブロックの中でバグが起きているコードを特定します。

④ バグを修正し，プログラムを実行して，意図した正しい実行結果が得られているかを確認します。

⑤ デバッグも自分の頭で考えることが基本です。ただし，自力での修正が難しく時間を要する場合には，ChatGPT や友人，担当教員にバグの原因と対策を遠慮しないで尋ねてください。

第 5 章　多様なアルゴリズムとプログラミング

例題 5.13*　プログラム dp_coin.ipynb の説明

##　関数 min_coin_change() の前半の説明：　請求額を支払うための最小のコイン枚数を求める

最小コイン使用枚数を収める動的テーブルをリスト List_dP で表しています。List_dP は，L6 の print 出力が示すように，1，3，5 円硬貨を使う順に更新されていきます。5 円硬貨での最小使用枚数の更新が終わった時点での最終の list_dp の最後の要素 list_dp[amount] に，この問題の最小コイン使用枚数が収められていることを理解してください。

L3，L4 は，請求額を支払うために必要な最小コイン枚数をコイン（koin）ごとに求める 2 重の for ループです。L3 で，koin は辞書から取り出すキーのコインで，固有の額面が定まっています。koin をまず固定して，L4 で，koin を開始値として請求額 amount まで 1 円の増分で順に取り出して計算額（支払額）paymt に代入することを繰り返します。L5 で動的リスト list_dp のインデックス paymt の要素として，最小の koin 枚数を選んで代入しています（後述）。内側の for ループの range で開始値が koin で始まるのは，計算額が koin の額面以上でなければ，koin が使えないからです。

最小コイン枚数を収める list_dp は，1 円，3 円，5 円硬貨の順で，最小使用枚数が求められて**メモ化**されています。ここで注意しなければならないことは，3 円硬貨での支払いでは，1 円硬貨での支払いを含むことが許されていることです。3 円硬貨だけでは計算額の支払いができないからです。5 円硬貨での支払いは，1 円，3 円硬貨での支払いを含みます。5 円硬貨だけは計算額の支払いができないからです。つまり，最初に取り出されたコイン 1 円では，このコインだけしか支払いに使えませんが，次に取り出したコイン 3 円，5 円からは，それ以前のコインも使って支払いができる，ということです。なお，最小使用枚数を求めるためには，現在のコインと以前のコインは，自由に組み合わせることができます。

実行結果には，最初に 1 円硬貨の最小使用枚数が，計算額 paymt の 1 円から 7 円まで順に求められて，list_dp が動的に埋められていく様子が出力されています。1 円での処理では list_dp は，初期化で第 0 要素が 0 であることを含めて，[0，1，2，3，4，5，6，7] で終わり，これが次の 3 円の処理に引き継がれます。3 円の処理では，paymt が 1 円から 2 円までは 1 円の結果をそのまま引き継ぎ，paymt が 3 円から 7 円までは 3 円硬貨と 1 円硬貨を使用して最小使用枚数が算出されています。最後の 5 円では，paymt が 1 円から 4 円までは 1 円と 3 円の結果をそのまま引き継ぎ，paymt が 5 円から 7 円までは 5 円硬貨と 1 円硬貨を使用して最小使用枚数が算出されています。

このように動的計画法では，問題を**分割**し，先に処理する小さな問題の計算結果を**メモ化**で記録して同じ計算を何度も行うことを避けると同時に，小さな問題の計算結果を**再利用**してより大きな問題を解くのに使用しています。すなわち，動的計画法のアルゴリズムでは，動的計画の**モデル化**，**分解と分割**，繰り返しの**パターン認識**，**再利用**などの計算論的思考の概念が用いられていることがわかります。

ここで最小使用枚数を算出するには，L5 のコードを必要とすることに注意してください。

```
list_dp[paymt] = min(list_dp[paymt], list_dp[paymt-koin] + 1)
```

このコードにより list_dp のインデックス paymt の要素（最小コイン枚数）を更新します。上式の右辺は，当初の要素 list_dp[paymt] と新たに koin を使ったときの要素 list_dp[paymt-koin] + 1 とで，要素の値の小さい方を選んで左辺に代入して，インデックス paymt の要素（最小枚数）を更新しています。list_dp[paymt-koin] + 1 で，+1 は koin を 1 枚支払いに使用した結果です。そうすると，新たな計算額は paymt−koin となりますので，その支払い枚数は list_dp[paymt-koin] となります。このとき，

list_dp[paymt-koin]は，すでに計算済みで**メモ化**されていますので，結果を**再利用**するだけでよいことに気づいてください。

具体例でこれを確かめてみましょう。List_dp の初期値は [0, inf, inf, inf, inf, inf, inf, inf] です。1 円硬貨による最初の支払いは，計算額 1 円に対して 1 枚となりますので，List_dp は [0, 1, inf, inf, inf, inf, inf, inf] となります。この最初の処理で，1 円硬貨を使うことによる枚数は，min(list_dp[paymt], list_dp[paymt-koin] + 1) から，min(list_dp[1], list_dp[1-1] + 1) で求められます。list_p[0] は，初期化で 0 となっています。そこで，min(float('inf'), 1) から，更新された list_dp[1] は 1 となります。3 円硬貨での支払いで，各 paymt に対して最小枚数を順次求めてみます。

3 円硬貨での支払いの第 1 行 [0, 1, 2, 1, 4, 5, 6, 7]　　# paymt 3 に対して 3 円が一枚
3 円硬貨での支払いの第 2 行 **[0, 1, 2, 1, 2, 5, 6, 7]**　　# paymt 4 に対して 3 円と 1 円が各一枚
・
・

計算額 3 円を 3 円硬貨で支払うことで，第 1 行の第 4 要素は 1 に，計算額 4 円を 3 円硬貨と 1 円硬貨で支払うことで，第 2 行の第 5 要素は 2 に更新されています。後者の場合，

list_dp[paymt] = min(list_dp[paymt], list_dp[paymt-koin] + 1) は
list_dp[4] = min(list_dp[4], list_do[4-3] + 1)
list_dp[4] = min(4, 2) から，list_dp[4] = 2

となります。5 円硬貨による支払いも同様に考えればよい。

5 円硬貨での支払いの第 1 行 [0, 1, 2, 1, 2, 1, 2, 3]　　# paymt 5 に対して 5 円が一枚
5 円硬貨での支払いの第 2 行 [0, 1, 2, 1, 2, 1, 2, 3]　　# paymt 6 に対して 5 円と 1 円が各一枚
5 円硬貨での支払いの第 3 行 [0, 1, 2, 1, 2, 1, 2, 3]　　# paymt 7 に対して 5 円一枚と 1 円が二枚

この第 3 行が，動的に変化する list_dp の最終形態となり，この最後の要素 3 が請求額 amount 7 円を支払うのに必要な最小コイン枚数（最適解）となります。

L3～ L7 の 2 重の for ループを抜け出すと，最小コイン枚数は list_dp[amount]) に入っていますので，L9 でこれを出力しています。L10 で，これを min_num_coins に代入しています。この最小コイン枚数は，L29 で戻り値の一つとして返します。

関数 min_coin_change() の後半の説明： 各コインの使用枚数を求める

以下で対象とする動的リスト list_dp は，5 円硬貨についての処理の最終行で L8 の出力です。計算額（支払額）と対応させて示すと下記となります。

```
# 計算額（支払額） 0, 1, 2, 3, 4, 5, 6, 7
the last list_dp [0, 1, 2, 1, 2, 1, 2, 3]
```

請求額 7 円を支払うのに必要な最小コイン枚数（最適解）は 3 で，これを満たすコインは 5 円が 1 枚と 1 円が 2 枚であることは，このような簡単なケースでは直ぐに分かります。list_dp[7] == list_dp[7-5] + 1 ですから，右辺第 1 項は list_dp[2]となりますが，これは 1 円硬貨が二枚使われたことを示します。右辺の第 2 項 +1 は，5 円硬貨が一枚使われたことを示しています。以下では，これをプログラムで求めることにします。

各コインの使用枚数は，コインの辞書 dict_coins に収めます。L32 でこれを初期化しています。L14 は計算額（支払額）patmt の初期化で，請求額 amount を代入しています。L15～L27 で，for 節は各コインの使用枚数を求める繰り返しで，while 節は一つの koin に対して使用枚数を求めるための繰り返しです。L15 で，コイン辞書から koin と vnum を取り出します。vnum には L32 のコイン辞書の初期化で 0 が代入されています。まず一つの koin を固定して，その koin が何枚使われたかを while 節で調べています。while 節の

繰り返しの条件式

```
(paymt >= koin) and (list_dp[paymt] == list_dp[paymt-koin] + 1)
```

は，2 つの条件式が and で結ばれています。and の左側の条件式は，計算額 paymt が koin の額面以上であることです。これが満足されていると，koin が 1 枚使えます。さらに and の右側の条件式が満足されていれば，koin の使用枚数をカウントできます。and の右側の条件式の == を挟んで右辺第 1 項は，計算額から koin の額面を引いた金額の最小コイン枚数，右辺第 2 項の +1 は，koin を 1 枚使ったので，枚数を +1 することを表しています。koin が 1 枚使われているとすると，list_dp[paymt] == list_dp[paymt-koin] + 1 が成り立つはずです。そこでこの等式が成り立てば，L21 で koin の使用枚数のカウントを +1 し，L23 で使った koin の額面だけ計算額を減らします。ここで while 節の終わりに来ていますので，while 文まで戻り，その koin がさらに使われているかを調べます。その koin が使われていれば，vnum と計算額を更新します。その koin が使われていなければ，while 節を抜け出して，L26 でその koin の使用枚数 vnum を辞書 dict_coins に収めます。これを終えると for 文に戻り，次の koin を取り出します。このときには計算額は前に使用した koin 枚数分だけ減らされて更新されています。while 節ではこの新しい koin が使われている枚数を同様にして調べ，koin の使用枚数を辞書に収め，計算額を更新します。koin がまったく使われていないときには，計算額は更新されず，vnum には初期化した 0 が入っています。以上の手続きを繰り返すことで，各コインの使用枚数を求めて，コイン辞書 dict_coins を完成させ，L29 で戻り値の一つとして返します。今の場合，7 円の請求額を計算額（支払額）の初期値として，最初に 1 円硬貨を 2 枚使用できますので，残りの計算額は 5 円となります。次の 3 円硬貨では 5 円の支払いはできません。最後の 5 円硬貨では，一枚使用してちょうど 5 円の支払いができ，残りの支払額は 0 となり，これで計算が完了します。L8 の出力 (the last coin 5)で，5 円が last coin であることを示しています。

このプログラムでは，コイン辞書の項目をキーの昇順に並べていますが，昇順でなくてもかまいません。

例題 5.15* プログラム dp_knpsk.ipynb の説明

 # 関数 dp_knapsack() の前半の説明： dp_table（2 次元のリスト dp）を作成し，最大価値を求める

dp_table を 2 次元リストで表すと，行は品番 0 から品番 n までの間隔値 1 の整数をとり，列は重さで 0 から WC までの間隔値 1 の整数をとります。リスト dp でセル dp[i][j] は，i 行 j 列の要素である価値の和を表します。以下では dp[i, j] に収める価値の和をどのようにして決めていくかを考えます。

関数 dp_knapsack()は，2 重の for ループから成り立っています。L3, L5 の 2 つの for 文の開始値は 1 からでよい。なぜなら初期化で 0 は済んでいるからです。まず i を固定して，j を 1 から WC まで +1 の増分で繰り返します。すなわち，価値の和を収めていく計算は，セル dp[1][1] から始まって行方向に dp[1][2], dp[1][3], . . . , dp[1][WC] まで終えたら，次の行の計算に移ります。これを n 行まで繰り返します。このようにして dp_table には価値の和がメモ化されていきます。

内側の for 節のブロックでやっていることは，i 番目の品物をナップザックに新たに入れることができるか否かを，品番 i の重さ wi が重量の上限 j を超えるかどうかで処理を分岐させる判断でしています。

wi > j ならば品番 i の品物は使えませんので，L6, L7 に示すように，dp[i][j] は，一つ上の行のセルである dp[i-1][j] を引き継いで，価値の和は増えず不変です。すなわち，dp_table の同一列で，dp[i-1][j] = dp[i][j] の場合には，品番 i の品物がザックに入れられないことを示しています。

wi <= j ならば，品番 i の品物を入れることができる場合があります。L9

```
dp[i][j] = max(dp[i-1][j], vi + dp[i-1][j-wi])
```

が示すように，セル dp[i][j]に収める価値は，一つ上の行のセルの dp[i-1][j] か，vi + dp[i-1][j-wi] のどちらか大きい方となります。ここで wi は 0 または正の整数をとることに注意（wi は 0 または正の整数でないと，ナップザック問題の動的計画法はうまくいきません。2 次元リストの index は 0 または正の整数でなければなりませんから)。後者は，品番 i の品物をザックに入れるとなると，新たな価値の和は vi に dp[i-1][j-wi] を加えたものになることをいっています。dp[i-1][j-wi] は，i-1 行で j-wi 列の価値を表します。価値 vi を加えるときには，同時に重さ wi を加えることになりますが，j-wi は，wi を加えても上限 j を超えないようにする許される残り重さです。

L9 が示すように，dp[i-1][j] > vi + dp[i-1][j-wi] であれば，dp[i][j] には，dp[i-1][j] が代入され，価値の和は増えず不変で，品番 i の品物をザックに入れることはできません。dp[i-1][j] ＜ vi + dp[i-1][j-wi] であれば，より大きな価値の和 vi + dp[i-1][j-wi] が dp[i][j] が代入され，品番 i の品物はザックに入れることができます。dp_table の同一列で，dp[i-1][j] < dp[i][j] で価値が増えた場合には，品番 i の品物がザックに入れられていることを示しています。この性質は，関数の後半のプログラムで，最大価値を得るために選んだ品物の辞書を作成する場合に使います。

以上の手続きを品番 1 から品番 n まで繰り返すことにより，セルに収められた価値が単調増加となる更新が行方向と列方向におこなわれ，最終的に dp[n][WC] に最大価値が収められます。そこで，L10 で dp[n][WC]を max_value に代入しています。上記の説明を (n+1) x (WC+1) の dp_table で確認してください。

 # 関数 dp_knapsack() の後半の説明： 最大価値を与える品物の辞書を作成する

品番 1 から品番 n(=5)までの品物の辞書は

dict_goods = {1:(4, 3), 2:(3, 2), 3:(6, 5), 4:(5, 3), 5:(1, 2)}

で与えられています。辞書のキーが品番，値が（価値，重さ）です。ナップザック問題の解としては，最大価値を求めるだけでなく，品番 1 から品番 n までの n 個の品物のうち，どの品物とどの品物を選んで最大価値に達したかを知る必要があります。これを品番をキー，値を（価値，重さ）とする辞書 sdict として求めることにします。

品番 i の品物をザックに入れることができる場合は，dp[i-1][j] < dp[i][j] が成り立つ場合です。dp[i-1][j] = dp[i][j] の場合には，品番 i の品物をザックに入れることはできません。この条件の下で，最適解（最大価値を求めるために選んだ品物の組み合わせ）を求める手続きは，最後にザックに入れた品物を定めることから始めます。すなわち，最大価値を求めた方向とは逆方向に，最大価値を与えるセル dp[n][WC]（今の場合は dp[5][10]）から始まり，dp_table を遡って，dp[i-1][j]と dp[i][j]の二つのセルで上の条件式が成り立つかどうかを調べて，ザックに入れた順序と逆順で，選んだ品物を定めて，辞書 sdict に代入していくことになります。

L13，L14 は，辞書と探索の開始行の初期化です。L14 で j = WC として，探索を最大列から始めます。L15～L20 の for 節の繰り返しでは，for 文の range() 関数は，開始値 n，終了値 0，間隔値-1 ですから，i は n から始まって n-1, . . . , 1 の値をとることになります。 以下，WC = 10 で j = 10 から, n = 5 で i= 5 から始めます。

for 節の繰り返しの最初は i = 5，j = 10 で，L18 の条件式 dp[4][10] < dp[5][10] が成り立つかどうかの判断をします。今の例ではこの条件式は成り立たず，品番 5 の品物は選ばれていませんので，for 文に戻ります。次の i = 4 ，j = 10 では dp[3][10] < dp[4][10] が成り立ちますので，品番 4 の品物が選ばれていることが分かります。そこでこれを辞書に代入します。L20 で，品番 4 の品物の重さ 3 を差し引きますので，新たな j は，7(= 10-3) となります。ここで，list_goods の index は，0 から始まることに注意してください。for 文に戻って i = 3 で，L18 の条件式 dp[2][7] < dp[3][7] が成り立ちますので，品番 3 の品物が選ばれていることが分かります。これを辞書に代入して， j を更新して j = 2(= 7-5)となり，for 文に戻って i = 2 で，dp[1][2] < dp[2][2] が成り立ちますので，品番 2 の品物が選ばれていること

が分かります。これを辞書に代入して，j が更新されて j = 0(= 2-2) となり，for 文に戻って i = 1 の最後の繰り返しとなりますが，L16，L17 により for 節が終了します。上記の説明を dp_table で確認してください。

以上の手続きにより，最適解となる選んだ品物の辞書 sdict が得られます。

sdict : {4: (5, 3), 3: (6, 5), 2: (3, 2)}

L35～L37 では，選んだ品物について価値の和と重さの和を求めて確認しています。

value sum: 14 weight sum: 10

この場合には重さの和は上限値 WC に等しくなりましたが，いつも重さの和は WC に等しいとは限りません。

L31～L33 の dp_table の出力は，WC の値が大きなときには表が巨大になりますのでコメントとしておくとよい。

品物の辞書では，キーを品番としてキーの昇順に項目を並べています，キーに対応させる辞書の値にどの品物の（価値，重さ）を割り当てるかは任意です。辞書での品物の並べ方により，動的テーブルでの価値の動的変化は異なりますが，最適解（最大価値）と選ばれた品物は同一となります。これを確認してください。

ナップザック問題の動的計画法でも，問題を**分割**し，先に処理する小さな問題の計算結果を**メモ化**で記録して同じ計算を何度も行うことを避けると同時に，小さな問題の計算結果を**再利用**してより大きな問題を解くのに使用していることに気づきましたか。

課題解答

2024．05.24 版

本 Web サイトの利用は，お客様ご自身の責任と判断によっておこなってください。本書（『計算論的思考を育む Python プログラミング実践問題集』）と本 Web サイトに掲載されている内容およびサンプルプログラムに基づくいかなる運用結果に関しても，著者ならびに弊社は一切の責任を負いかねます。

課題プログラムについて，回答例を示しています。

・ソースコードには，インラインコメントを付けています。
・一部の課題については，プログラムの短い説明をしたり，実行結果は示したりしています。
・実行結果を示していないプログラムについては，プログラムの実行を読者にお任せしています。計算結果を是非自らご確認ください。
・プログラムの実行に際しては，別フォルダに収載の ファイル（拡張子 .ipynb）をご利用ください。

第 1 章手続き型プログラミング入門

1.1　順次構造と順次処理，プログラムのフラット化

課題 1.1

```
j10, j7, j3 = [10, 0, 0]          # 初期状態
j10, j7, j3 = [3, 7, 0]
j10, j7, j3 = [3, 4, 3]
j10, j7, j3 = [6, 4, 0]
j10, j7, j3 = [6, 1, 3]
j10, j7, j3 = [9, 1, 0]
j10, j7, j3 = [9, 0, 1]
j10, j7, j3 = [2, 7, 1]
j10, j7, j3 = [2, 5, 3]
j10, j7, j3 = [5, 5, 0]          # 最終状態
print(j10, j7, j3)
```

不定方程式 $7m + 3n = 5$ において，これを満たす整数を求めれば $m = 2, n = -3$ となります。これから，7L の Jag を 2 回満たし，3L の Jug を 3 回空にすればよいことが分かります。

不定方程式 $7m + 3n = 5$ において，これを満たす整数には $m = -1, n = 4$ もありますので，もう一つの解があります。

もう一つの解

```
j10, j7, j3 = [10, 0, 0]          # 初期状態
j10, j7, j3 = [7, 0, 3]
j10, j7, j3 = [7, 3, 0]
```

```
j10, j7, j3 = [4, 3, 3]
j10, j7, j3 = [4, 6, 0]
j10, j7, j3 = [1, 6, 3]
j10, j7, j3 = [1, 7, 2]
j10, j7, j3 = [8, 0, 2]
j10, j7, j3 = [8, 2, 0]
j10, j7, j3 = [5, 2, 3]
j10, j7, j3 = [5, 5, 0]            # 最終状態
print(j10, j7, j3)
```

課題 1.2

```
total = 43200
x = total / sum(range(1, 9))
print('1L の鍋の値段 ', x)                              # 1200.0
listp = [x, 2*x, 3*x, 4*x, 5*x, 6*x, 7*x, 8*x]     # 要素は鍋の値段
print(sum(listp))                                      # 組込み関数 sum(), 確認
```

課題 1.3＊

```
x = int(input('Enter a money(yen): '))      # 10000
r = 10                                        # 10 日で 10%
n = 365/10                                    # 1 年の回数
sums = x + x * r /100 * n
print('単利の場合の元利合計（円）', int(sums))      # 46500
sumc = x * (1 + r/100)**n
print('複利の場合の元利合計(円）', int(sumc))       # 324214
```

課題 1.4

```
import math
x, y = map(int, input('Enter integers with a spce: ').split())
print('GCD:', math.gcd(x, y))
```

課題 1.5

```
from sympy import isprime                          # ライブラリのインポート
n = int(input('Enter a natural number : '))
print(isprime(n))                                   # True/False の判定もする
```

True/False の判定を **if** 文で条件分岐を使うことなく **print()** 関数でして出力させています。これを **if,**

else 節を使って，

```
if isprime(n):
  print(True)
else:
  print(False)
```

とすることもできますが，プログラムはフラットになりません。

課題 1.6

```
test = ['Math', 65, 73.6, 98.5, 68, 54, 75.8]
maxv = max(test[1:])
minv = min(test[1:])
mean = round(sum(test[1:])/len(test[1:]), 2)    # 小数点以下 2 桁まで四捨五入する
print(maxv, minv, mean)                          # 98.5 54 72.48
```

課題 1.7

```
listx = [int(i) for i in input().split()]        # 内包表記。たとえば，1 2 3
print(listx)
print(listx.count(listx[0]) == len(listx))       # True/False の判定
```

count(listx[0]) は，リストのインデックスが 0 の要素がいくつあるかをカウントします。他に，組み込み関数 all() を使うこともできます。たとえば

```
listx = [20, 20, 20]
print(all(i == listx[0] for i in listx ))        # True
```

if else 節を使うことでもできますが，冗長となります。

```
listx = [int(i) for i in input().split()]
print(listx)
setx = set(listx)                            # リストから集合を作る
if len(setx) == 1:
  print('all elements are equal', setx)
else:
  print('not equal')
```

ここでは，重複する要素は持てないという集合の性質を使っています。

課題 1.8

```
import random                                                    # random モジュールのインポート
n = 16                                                           # パスワードの文字数
chas = 'あいうえおかきくけこさしすせそたちつてとなにぬねの\
はひふへほまみむめもやゆよらりるれろわゐゑをん'
list_pwd = [random.choice(chas) for i in range(n)]     # 内包表記
print(list_pwd)
str_pwd = ''.join(list_pwd)                                      # メソッド
print(str_pwd, len(str_pwd), sep = '\t')                         # パスワード
```

課題 1.9

```
datax = 'abcdefg'
print('元のデータ ', datax)
#1 文字列をスライスする
print('逆順 1', datax[::-1])
#2 組み込み関数  reversed(data) を使う
print(reversed(datax))                                           # イテレータの生成
print(list(reversed(datax)))                                     # リストをつくる
print('逆順 2', ''.join(list(reversed(datax))))
print(datax)                                                     # 元のデータは変更されていない
```

文字列型には逆順にするメソッド reverse() がない理由は，文字列型はイミュータブルですので，元の文字列を破壊して逆順にすることができないからです。

課題 1.10＊

```
import random
random.seed(200)                                         # デバッグを考えて seed()を設定し固定
list_test = [random.randint(0, 100) for i in range(20)]     # 内包表記 1
print(len(list_test), list_test, )
list_ds = sorted(list_test, reverse = True)      # 要素の並びを降順にする
list_pass = [i for i in list_ds if i >= 60]                      # 内包表記 2
print(len(list_pass), list_pass)
```

繰り返しを内包表記 1，条件判断を内包表記 2 の中に入れることにより，順序処理のプログラムとしています。内包表記 2 を使わず，合否を if　else 節で決めるプログラムを作ることもできますが，

```
list_pass = []
for i in list_ds:
  if i >= 60:
    list_pass.append(i)
```

となり。プログラムはフラットにならず，複雑になります。内包表記 2 の代わりに，高階関数の filter() と lambda 式を使うこともできます。

```
list_pass = list(filter(lambda x : x >= 60, list_ds))
```

これではかえって難しく感じるかもしれませんが，ここでは関数を使えばプログラムがフラットになることを理解することでかまいません。

課題 1.11＊

```
import pprint
list_rec = [('Mary', 'B', 75), ('David', 'A', 85,), ('Suzan', 'C', 65),
            ('John', 'S', 95), ('Robert', 'F', 55), ('Alice', 'A', 80)]
list_name = sorted(list_rec, key = lambda x: x[0])                    # 名前の昇順
pprint.pprint(list_name)
print()
list_name = sorted(list_rec, key = lambda x: x[2], reverse = True)   # 点数の降順
pprint.pprint(list_name)
```

課題 1.12＊

```
import numpy as np
x = [83, 79.5, 50, 68, 72, 60.5, 90, 85, 55, 95]       # eng，英語の点数
y = [85, 74, 55, 72.5, 64, 60, 81, 77, 63.5, 90]       # jpn，国語の点数
meanx = np.mean(x)
meany = np.mean(y)
stdx = np.std(x)
stdy = np.std(y)
covarxy = 0
for i, j in zip(x, y):
  covarxy += (i - meanx) * (j -meany)
covarxy = covarxy / len(x)
corr = covarxy / (stdx * stdy)
print('相関係数 r：', round(corr, 3))        # 0.927
a = corr * stdy / stdx
b = meany - a * meanx
print('傾き a：', round(a, 3))                 # 0.694
print('y 切片 b：', round(b, 3))               # 20.956

print('\n', 'Use Numpy.')
print('相関係数 r：', round(np.corrcoef(x, y)[0, 1], 3))
a, b = np.polyfit(x, y, 1)
print('傾き a：', round(a, 3))
```

```
print('y 切片 b：', round(b, 3))
```

1.2　条件分岐構造と条件分岐処理

課題 1.13

```
bmi = float(input('Enter a BMI: '))
if bmi <= 18.0 :
  print('ファッションショーへの出演不可')
else:
  print('ファッションショーへの出演可')
```

if 文と else 文は同格です。

課題 1.14

```
height = float(input('身長（cm）を入力してください:  ')) / 100
weight = float(input('体重（kg）を入力してください:  '))
bmi = weight / height ** 2
print('BMI', round(bmi, 1))
if bmi < 18.5:
  print('低体重です')
elif 18.5 <= bmi < 25.0:
  print('普通体重です')
elif 25 <= bmi < 30.0:
  print('肥満（1 度）です')
elif 30 <= bmi < 35.0:
  print('肥満（2 度）です')
elif 35 <= bmi < 40.0:
  print('肥満（3 度）です')
else:
  print('肥満（4 度）です')
```

課題 1.15＊

```
year = int(input('Enter a year: '))
if (year % 400 == 0) or (year % 4 == 0 and year % 100 != 0):
  print( year, ' True, a leap year' )               # 規則 (1)
else:
  print( year, ' False, a common year' )            # 規則 (2)
```

別解

```
from calendar import isleap                # モジュールのインポート
year = int(input('Enter a year = '))
print(isleap(year))                         # 関数の呼び出し
```

課題 1.16

```
string1 = input('Enter a string: ')
string2 = input('Enter a string: ')
print(string1, len(string1), sorted(string1))
print(string2, len(string2), sorted(string2))
if len(string1) != len(string2) or sorted(string1) != sorted(string2):
  print('not anagram')
else:
  print('anagram')
```

別解

```
import itertools
string1 = input('Enter a string: ')
string2 = input('Enter a string: ')
ang_list = [''.join(w) for w in itertools.permutations(string1)]
print('anagram list', ang_list)
if string2 in ang_list:
  print('anagram')
else:
  print('not anagram')
```

ここで,

`itertools.permutations(iterable, r=None)`

は，iterable の要素からなる長さ r の順列 (permutation) を連続的に返します。r が指定されない場合にはデフォルトで iterable の長さとなり，可能な順列の全てが生成されます。

課題 1.17

```
w = float(input('Enter a weight(g) = '))
if w <= 25.0:
  postage = 84
elif 25.0 < w <= 50.0:
  postage = 94
if 0 < w <= 50:
  print('郵便料金は', postage, '円です。')
else:
  print('定形郵便物ではありません。')
```

課題 1.18

```
w = float(input('Enter a weight(g): '))
if 0 < w <= 50:
  postage = 120
elif 50 < w <= 100:
  postage = 140
elif 100 < w <= 150:
  postage = 210
elif 150 < w <= 250:
  postage = 210
elif 250 < w <= 500:
  postage = 390
elif 500 < w <= 1000:
  postage = 580
if 0 < w <= 1000:
  print('郵便料金は', postage, '円です。')
else:
  print('郵便物は規格外です。')
```

1.3 繰り返し構造と繰り返し処理

課題 1.19

```
n = int(input("Enter a number: "))
for i in range(n, -1, -1):
  print(i)                                # n, n-1, . . ., 1, 0
```

課題 1.20

```
sumx = 0
for i in range(1, 11):
  sumx += 9/pow(10, i)
  print(i, ': ', sumx)
```

課題 1.21

```
from math import factorial
sumx = 0
for i in range(1, 11):
  sumx += i/factorial(i+1)
```

```
  print(i, ': ', sumx)
```

課題 1.22

```
from math import factorial
sumx = 0
for i in range(1, 12):
  sumx += i/factorial(i)
  print(sumx)
```

課題 1.23

```
from math import factorial
sumx = 1
for i in range(1, 11):
  sumx += i*i/factorial(i+1)
  print(sumx)
```

課題 1.24

```
teams = ['A', 'B', 'C', 'D', 'E', 'F']
oppnts = ['A', 'B', 'C', 'D', 'E', 'F']
print('参加チーム： ', len(teams))
num = 0
dict_league = {}        # {'A':'' , 'B':'' , 'C':'' , 'D':'' , 'E':'' , 'F':''}
for i in teams:
  oppnts.remove(i)
  print('チーム', i, '  相手チーム', tuple(oppnts))
  num += len(oppnts)
  dict_league[i] = tuple(oppnts)
print('試合総数:', num)
print(dict_league)
```

1.4　繰り返しと条件処理の組み合わせ

課題 1.25

```
import string
stringx = '2 * 3 + 4 / 5'
list_eqn = stringx.split()
print(list_eqn)                          # ['2', '*', '3', '+', '4', '/', '5']
```

```
for i, elm in enumerate(list_eqn):      # リストのインデックスと要素の 2 つを取り出す
  if elm not in '+-*/':
    list_eqn[i] = int(elm)
print(list_eqn)                          # [2, '*', 3, '+', 4, '/', 5]
```

課題 1.26

```
for i in range( 2, 8):
  n = 101**i
  print('101 の', i, '乗')
  print(str(n))
  if str(n) == str(n)[::-1]:
    print(True)
    print()
  else:
    print(False)
    print()
```

課題 1.27

```
n = 200
for i in range(1, n+1):
  i = i * 11
  if i == 110:
    continue
  if i > 200:
    break
  if i % 11 == 0:
    j = str(i)
    print(j)
    if int(j[::-1]) % 11 == 0:
      print(int(j[::-1]))
      print(True)
      print()
```

課題 1.28

```
n = 50
for i in range(3, n+1, 3):       # 3 の倍数
  j = str(i)                      # 数字列に変換
  sumx = 0
  for k in j:                     # 各桁の数字を取り出す
```

```
    sumx += int(k)                  # 整数に変換
  print(i, ' digit sum', sumx, end = '  ')
  if sumx % 3 == 0:                 # 各桁の数の和が 3de 割り切れるか
    print(True)
```

課題 1.29

```
n = 200
for i in range(9, n+1, 9):              # 9 の倍数
  j = str(i)
  sumx = 0
  for k in j:
    sumx += int(k)
  print(i, ' digit sum', sumx, end = '  ')
  if sumx % 9 == 0:
    print(True)
```

課題 1.30

```
months = {1:'January', 2:'February', 3:'March', 4:'April', 5:'May', 6:'June',
7: 'July', 8:'August', 9:'September', 10:'October', 11:'November', 12:'December'}
print('Take oysters in the following months.')
for k, v in months.items():
  if 'r' in v:
    print(k, '月',  '\t', v)
```

課題 1.31

```
text = 'Peter Piper picked a peck of pickled peppers;\
A peck of pickled peppers Peter Piper picked;\
If Peter Piper picked a peck of pickled peppers,\
Where’s the peck of pickled peppers Peter Piper picked?'
countx = 0
for i in text:
  if i == 'P' or i == 'p':
    countx += 1
print('P と p の数:', countx)
print('P の数:', text.count('P') )      # 大文字と小文字を区別する
print('p の数:', text.count('p'))
```

課題 1.32

```
year = {'1月':31, '2月':28, '3月':31, '4月':30, '5月':31, '6月':30,
        '7月':31, '8月':31, '9月':30, '10月':31, '11月':30, '12月':31}
large = []                                  # 大の月を入れるリストの初期化
small = []                                  # 小の月を入れるリストの初期化
for m, d in year.items():                   # キーと値のペアを一つずつ取り出す
  if d == 31:                               # 大の月
    large.append(m)                         # リストの末尾に追加していく
  else:                                     # 小の月
    small.append(m)
print('大の月', large)
print('小の月', small)
```

別解： 大小の月の特徴を掴んでプログラムすると，次のプログラムとなります。

```
for month in range(1, 13):                  # 1, 2, 3, . . . , 11, 12
  if month % 2 == 0 and  month >= 8:
    print(month, ' 大の月')
  elif month % 2 == 0 and  month < 8:
    print(month, ' 小の月')
  elif month % 2 != 0 and  month >= 9:
    print(month, ' 小の月')
  elif month % 2 != 0 and  month < 9:
    print(month, ' 大の月')
```

プログラムは，実行時間に大きな違いがなければ，多少コード数が増えても分かりやすいプログラムとするほうがよい。

課題 1.33＊

```
import hyakunin_isshu
letter_group = [['む', 'す', 'め', 'ふ', 'さ', 'ほ', 'せ'],
                ['う', 'つ', 'し', 'も', 'ゆ'],['い', 'ち', 'ひ', 'き']]
for i in letter_group:
  num = []
  for j in i:
    for k, v in hyakunin_isshu.hyakunin_isshu.items():
      if j == v[0]:
        num.append(k)
        print(k, v)
  print(sorted(num), len(num))
  print()
```

課題 1.34

```
num = 1
while num <= 20:
  if num % 3 == 0 and num % 5 == 0:
    print('Fizz&Buzz')
  elif num % 3 == 0:
    print('Fizz')
  elif num % 5 == 0:
    print('Buzz')
  else:
    print(num)
  num += 1
```

課題 1.35＊

```
n = int(input('Enter an integer n = '))            # たとえば, n = 20
print('3 の倍数 Fizz：', list(filter(lambda x: x % 3 == 0, range(1, n+1))))
print('5 の倍数 Buzz：', list(filter(lambda x: x % 5 == 0, range(1, n+1))))
print('両方の倍数 FizzBuzz：',
      list(filter(lambda x:  x % 3 == 0 and x % 5 == 0, range(1, n+1))))
```

課題 1.36＊

```
set_F, set_B, set_FB = set(), set(), set()
for num in range(1, 21):
  if num % 3 == 0 and num % 5 == 0:
    set_FB.add(num)
  elif num % 3 == 0:
    set_F.add(num)
  elif num % 5 == 0:
    set_B.add(num)
print('set of Fizzes: ', set_F, len(set_F))           # {3, 6, 9, 12, 18} 5
print('set of Buzzes: ', set_B, len(set_B))           # {10, 20, 5} 3
print('set of FizzBuzzes: ', set_FB, len(set_FB))    # {15} 1
```

課題 1.37＊

```
set_all = {i for i in range(1, 17)}
set_3 = {i for i in range(1,17) if i % 3 == 0}
set_5 = {i for i in range(1, 17) if i % 5 == 0}
set_f = {i for i in range(1, 17) if i % 3 == 0 and i % 5 != 0}
set_b = {i for i in range(1, 17) if i % 3 != 0 and i % 5 == 0}
```

```
set_fb = {i for i in range(1, 17) if i % 3 == 0 and i % 5 == 0}
set_oth =  {i for i in range(1, 17) if i % 3 != 0 and i % 5 != 0}
print('set_all:  ', len(set_all), set_all)
print('set_3:  ', len(set_3), set_3)
print('set_5:  ', len(set_5), set_5)
print('set_f:  ', len(set_f), set_f)
print('set_b:  ', len(set_b), set_b)
print('set_fb:  ', len(set_fb),set_fb)
print('set_oth:  ', len(set_oth),set_oth)
print()
print(set_f | set_b | set_fb | set_oth)
print(set_3 | set_5 | set_oth)
print(set_3 & set_5)
print(set_3 - set_fb)
print(set_5 - set_fb)
print(set_f ^ set_b)
```

1.5　総合問題

課題 1.38

```
import pprint
listx = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
listp = [1/36, 2/ 36, 3/36, 4/36, 5/36, 6/36, 5/36, 4/36, 3/ 36, 2/36, 1/36]
dictxp = dict(zip(listx, listp))     # 辞書。キー 目の数の和：値 生起確率
print(len(dictxp))
pprint.pprint(dictxp)
ex = 0
for i, j in dictxp.items():           # for i, j in zip(listx, listp): でも可
  ex += i * j
print('期待値 ', round(ex, 1))        #  7.0
```

最大の生起数は 6，生起確率は　6/36 = 0.167 。このときの目の数の和は 7。ということで，「目の数の和 7 に賭けるのが最も有利」ということになります。

課題 1.39

```
listx = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
listp = [1/36, 2/ 36, 3/36, 4/36, 5/36, 6/36, 5/36, 4/36, 3/ 36, 2/36, 1/36]
dictxp = dict(zip(listx, listp))     # 辞書。キー 目の数の和：値 生起確率
print(round(dictxp[7], 3))            # 目の数の和が 7 の確率　0.167
prob_m7 = 0
for k in dictxp.keys():
```

```
  if k >= 7:
    prob_m7 += dictxp[k]
print(round(prob_m7, 3))                # 目の数の和が 7 以上の確率  0.583
prob_even = 0
for k in dictxp.keys():
  if k % 2 == 0:
    prob_even += dictxp[k]
print(round(prob_even, 3))               # 目の和が偶数の確率 1/2 = 0.5
```

課題 1.40

```
import numpy as np
from matplotlib import pyplot as plt
x = [1, 2, 3, 4, 5, 6]                  # x 座標
y = [1, 2, 3, 4, 5, 6]                  # y 座標
X, Y = np.meshgrid(x, y)
plt.scatter(X, Y, s = 80, c = 'k')  # 格子点を描く。格子点を大きく，色を黒にした
plt.grid()                               # 格子
for i in range(1, 7):                   # 第 2 対角線を含む，左上半分の 6 本の直線を描く
  x = list(range(1, i+1))
  y = list(range(7-i, 7))
  plt.plot(x, y)
for i in range(2, 7):                   # 第 2 対角線を含まない，右下半分の 5 本の直線を描く
  x = list(range(i, 7))
  y = list(range(1, 8-i))
  plt.plot(x, y)
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.show()
```

課題 1.41

```
import pprint
listx = [0, 1, 2, 3, 4, 5]                          # サイコロの目の差
listp = [6/36, 10/ 36, 8/36, 6/36, 4/36, 2/36]
ex = 0
for i, j in zip(listx, listp):
  ex += i * j
print('期待値 ', round(ex, 2))                        # 定義通り。  1.94
```

課題 1.42

```
listx = [0, 1, 2, 3, 4, 5]                          # サイコロの目の差
listp = [6/36, 10/ 36, 8/36, 6/36, 4/36, 2/36]
dictxp = dict(zip(listx, listp))
print(round(dictxp[2], 3))                          # 目の差が 2  の確率 8/36 = 0.222
prob_m3 = 0
for k in dictxp.keys():
  if k >= 3:
    prob_m3 += dictxp[k]                            # 目の差が 3 以上の確率 12/36 = 0.333
print(round(prob_m3, 3))
prob_odd = 0                                         # 目の差が奇数の確率 1/2 = 0.5
for k in dictxp.keys():
  if k % 2 != 0:
    prob_odd += dictxp[k]
print(round(prob_odd, 3))
```

課題 1.43

move3 と move4 の分岐のみが変更となりますので，新しいプログラムは例題 1.20 のプログラムの L10, L12，L13，L15 をそれぞれ次のように変更するだけでよい。

```
 # move 3: fw, Move with the wolf to the RB
 state.append((1, 0, 1,1))                          ## state3, C-FGW
# move4: fg, Move with the goat to the LB
 state.append((0, 0, 0, 1) )                        ## state4, FCG-W
```

課題 1.44＊

```
poleA = [4, 3, 2, 1]            # Pole A の初期状態
poleB = []                       # Pole B の初期状態
poleC = []                       # Pole C の初期状態
print('Pole A  ', poleA)
print('Pole B  ', poleB)
print('Pole C  ', poleC)
print()
poleB.append(poleA.pop())      # 1 回目の円盤の移動
poleC.append(poleA.pop())      # 2
poleC.append(poleB.pop())      # 3
poleB.append(poleA.pop())      # 4
poleA.append(poleC.pop())      # 5
poleB.append(poleC.pop())      # 6
poleB.append(poleA.pop())      # 7
poleC.append(poleA.pop())      # 8
poleC.append(poleB.pop())      # 9
poleA.append(poleB.pop())      # 10
```

```
poleA.append(poleC.pop())      # 11
poleC.append(poleB.pop())      # 12
poleB.append(poleA.pop())      # 13
poleC.append(poleA.pop())      # 14
poleC.append(poleB.pop())      # 15  円盤の最終の移動
print('Pole A  ', poleA)       # Pole A の最終状態の出力 []
print('Pole B  ', poleB)       # Pole B の最終状態の出力 []
print('Pole C  ', poleC)       # Pole C の最終状態の出力 [4, 3, 2, 1]
```

第 2 章　構造化プログラミング

2.1　条件分岐処理と繰り返し処理

課題 2.1

```
mgsq = [[2, 9, 4], [7, 5, 3], [6, 1, 8]]
sum_r0 = sum(mgsq[0])
sum_r1 = sum(mgsq[1])
sum_r2 = sum(mgsq[2])
print('列和', sum_r0, sum_r1, sum_r2)
sum_c0 = sum([mgsq[0][0], mgsq[1][0], mgsq[2][0]])
sum_c1 = sum([mgsq[0][1], mgsq[1][1], mgsq[2][1]])
sum_c2 = sum([mgsq[0][2], mgsq[1][2], mgsq[2][2]])
print('行和', sum_c0, sum_c1, sum_c2)
sum_d0 = sum([mgsq[0][0], mgsq[1][1], mgsq[2][2]])
sum_d1 = sum([mgsq[0][2], mgsq[1][1], mgsq[2][0]])
print('対角線和', sum_d0, sum_d1)
```

課題 2.2

```
pw = input('半角英数字のパスワードを入力 (8 文字以上 15 文字以下)。 ')
pw_len = len(pw)
if pw.isalnum() and pw[0].isupper() and (not pw.isalpha()) and\
(8 <= pw_len <= 15):
  print(pw)
else:
  print('入力エラー：パスワードは最初の文字が英大文字で，英字と数字が混在させ，\
8 文字以上 15 文字以下にしてください')
```

上のプログラムでは，パスワードが不正なときに何が原因で不正かを特定して表示できません。次のプログラムは **if～else** 節を入れ子にして，不正の種類ごとに不正の原因を表示するものです。この方がプログラ

ムを実行するだけのユーザーには親切ですが，書く方は **if～else** 節が入れ子の分岐になって面倒なプログラムとなります。

```
pw = input('半角英数字のパスワードを入力(8 文字以上 15 文字以下)： ')
pw_len = len(pw)
if pw.isalnum():                          # 英数字ならば
  if pw[0].isupper():                     # 最初の文字が大文字ならば
    if not pw.isalpha():                  # すべてが英字でないならば
      if 8 <= pw_len <= 15:
        print('適格なパスワードです：',  pw)
      else:
        print('入力エラー：8 文字以上 15 文字以下にしてください')
    else:                                 # すべて英字ならば
      print('入力エラー：数字を入れてください')
  else:
    print('入力エラー：最初の文字は英大文字にしてください')
else:
  print('入力エラー：記号を入れないで英数字のみとしてください')
```

課題 2.3

```
import random
import string
#random.seed(200)
pwd = ''                                          # パスワードの文字列
for cha in range(4):
  pwd += random.choice(string.ascii_uppercase)
  pwd += random.choice(string.ascii_lowercase)
  pwd += random.choice(string.digits)
  pwd += random.choice(string.punctuation)
print(pwd, len(pwd), sep = '\t')
list_pwd = random.sample(pwd, len(pwd))         # 要素の並びををランダム化したリスト
print(list_pwd)
str_pwd = ''.join(list_pwd)                # リストの要素の文字を連結した文字列パスワード
print(str_pwd, len(pwd), sep = '\t')
if str_pwd[0] in string.ascii_uppercase:            # 第 0 要素が大文字ならば
  print('先頭が大文字の長さ', len(str_pwd), 'のパスワード:', str_pwd)
else:                                                 # 第 0 要素が大文字でなければ
  pwd0 = random.choice(string.ascii_uppercase)      # 大文字を 1 字選ぶ
  repl_cha = random.choice(str_pwd)                  # 差し替える 1 つの字種
  pwdU = pwd0 + str_pwd.replace(repl_cha, '', 1)    # 差し替える 1 つの字種を削除
  print('先頭が大文字の長さ', len(pwdU), 'のパスワード:', pwdU)
```

課題 2.4

解答省略

課題 2.5＊

```
loan = int(input('借入金額（万円）を入力してください： '))
yrate = float(input('年利%）を入れてください： '))
years = int(input('返済年数をいれてください： '))
loan = loan * 10000
mrate = (yrate / 12) / 100
n = years * 12
bp = loan // n
print(bp)
print('回数', '返済額 = 元金 + 利息', '借入残高', sep = '\t')
sumi = 0.
sump = 0.
sumbp = 0.
for i in range(1, n + 1):
  ip = int(loan*mrate)
  xp = bp + ip
  loan = loan - bp
  sumi += ip
  sump += xp
  sumbp += bp
  print(i, xp, bp, ip, loan, sep = '\t')

print('回数', '返済額 = 元金 + 利息', '借入残高', sep = '\t')
xp = xp + loan
bp = bp +loan
sumbp = sumbp + loan
sump = sump +loan
loan = 0
print(n, xp, bp, ip, loan, sep = '\t')
print('切り捨ての場合で，最終回での借入残高を 0 にするために，返済額を修正しています')
print()
print('      利息の総額 ', int(sumi))
print('均等返済額の総額', int(sumbp))
print('    返済額の総額', int(sump))
```

課題 2.6＊

```
el_result = {'A': 340000,'B':280000, 'C': 160000, 'D':60000, 'E':15000}
nvalues = list(el_result.values())
```

```
num = []
for v in nvalues:
   for i in range(0, 7):
      num.append(v // (i+1))
numA = num[0:7]
print(‘A’, numA)
numB = num[7:14]
print(‘B’, numB)
numC = num[14:21]
print(‘C’, numC)
numD = num[21:28]
print(‘D’, numD)
numE = num[28:35]
print(‘E’, numE)
print()
num_sorted = sorted(num, reverse = True)
num_sort = num_sorted[0:7]
print(‘商の大きい順’, num_sort)
seatA = 0
seatB = 0
seatC = 0
seatD = 0
seatE = 0
for v in num_sort:
  if v in numA:
    seatA += 1
  elif v in numB:
     seatB += 1;
  elif v in numC:
     seatC += 1
  elif v in numD:
    seatD += 1
  elif v in numE:
     seatE += 1
print('seatA =', seatA)
print('seatB =', seatB)
print('seatC =', seatC)
print('seatD =', seatD)
print('seatE =', seatE)
```

課題 2.7 '

```
bottles = 5
while bottles > 0:
  if bottles == 1:
```

```
    print(bottles, 'bottle of beer on the wall')
  else:
    print(bottles, 'bottles of beer on the wall')
  if bottles == 1:
    print(bottles, 'bottle of beer')
  else:
    print(bottles, 'bottles of beer')
  print('Take one down, pass it around')
  bottles = bottles - 1
  if bottles == 0:
    print('no more bottle of beer on the wall', '\n')
  elif bottles == 1:
    print(bottles, 'bottle of beer on the wall', '\n')
  else:
    print(bottles, 'bottles of beer on the wall', '\n')
```

課題 2.8

```
for i in range(1, 10):
  num10 = i / 10
  print(num10)
  num = num10
  listx = []
  i = 0
  while num != 0:                    # 無限ループ
    listx.append(int(num*2))
    num = num*2 - int(num*2)
    i += 1
    if i >= 31:                       # 無限ループからの脱出
      break
  print(listx)
  strx = ''.join([str(i) for i in listx])
  print(num10, 'を 2 進記数法で表すと  ', '0.' + strx)
```

循環小数にならない 10 進小数は，0.5 です。

2.2　関数の自作とモジュール化

課題 2.9＊

```
def total(oe):                  # 関数定義
  sumx = 0
  for x in range(oe, 101, 2):
```

```
    sumx += x
  return sumx

odd = 1
sumo = total(odd)              # 奇数の和  2500
even = 2
sume = total(even)             # 偶数の和  2550
print('奇数の和：', sumo, '  偶数の和：', sume, '  総和：', sumo + sume,)
```

課題 2.10

```
def mgsquare(a, c, g) :
  CNST = 15
  e = 5
  i = CNST - a - e
  b, f, h, d = CNST - a - c, CNST - c - i, CNST - g - i, CNST - a - g
  mgsq = [[a, b, c], [d, e, f], [g , h, i]]
  return mgsq

import pprint
for a in [2, 4, 6, 8] :
  if a == 2 or a == 8:
    c, g = 4, 6
    pprint.pprint(mgsquare(a, c, g), width = 20)
    print()
    pprint.pprint( mgsquare(a, g, c), width = 20)
    print()
  elif a == 4 or a == 6:              # else:  でもよいが，elif を使うほうが明瞭
    c, g = 2, 8
    pprint.pprint(mgsquare(a, c, g), width = 20)
    print()
    pprint.pprint(mgsquare(a, g, c), width = 20)
    print()
```

課題 2.11＊

```
def get_Chi2(L, N):
  dict_num = {i: 0 for i in range(L)}     # 辞書の初期化
  for i in range(N):
    x = random.randrange(10)
    dict_num[x] += 1
  F = N / L                                  # 理論度数
  Chi2 = 0                                   # Chi2 乗値の初期化
  sumf = 0
```

```
  for i in range(L):
    n = dict_num[i]                          # 実現度数（観測度数
    Chi2 += (n - F)**2 / F                   # カイ 2 乗値を求める
    sumf += n
  return dict_num, Chi2, sumf

import random
random.seed(100)
L, N = 10, 20000
limit = 16.919                       # 自由度 L - 1, 有意水準 0.05  で，境界値 16.919
for i in range(1, 11):               # 10 回の試行
  dict_num, Chi2, sumf = get_Chi2(L, N)
  print(i, '  Chi2 値 : ', round( Chi2, 4))
  print('辞書 : ', dict_num, 'サイズ', len(dict_num), '  総和 : ', sumf,
        '  理論度数 : ', N / L)
  if Chi2 < limit:
    print('0 から 9 の整数は，等しい確率で現れているといえる。\n')
  else:
   print('0 から 9 の整数は，等しい確率で現れているとはいえない。\n')
```

if　elif節の条件分岐を長々と書く代わりに，辞書をいつも用いることができるとは限らないことに注意。

課題 2.12

```
import random                            # random モジュールのインポート
def dice(n):
  num1 = [0] * 7                         # nim1[0], num2[0] は埋め草
  num2 = [0] * 7
  X = [0] * 13                           # X[0] , X[1] は埋め草
  st_prob = [0] * 13                     # st_prob[0] st_prob[1] は埋め草
  for i in range(1, n+1):
    x = random.randint(1, 6)
    y = random.randint(1, 6)
    num1[x] = num1[x] + 1
    num2[y] = num2[y] + 1
    X[x + y] = X[x + y] + 1
  for j in range(2, 13):
    st_prob[j] = round(X[j] /n, 4)
  return st_prob[2:]                     # st_prob[2], st_prob[3], . . . , st_prob[12]

n = 6000
limit = 600000
while True:
  print('統計的確率 ', 'n =', n)
  stat_prob = [round(i, 4) for i in dice(n)]
```

```
  print(stat_prob, len(stat_prob))
  n = n * 10
  if n > limit:
    break
print('先験的確率 ')
apri_prob = [1/36, 2/36, 3/36, 4/36, 5/36, 6/36, 5/36, 4/36, 3/36, 2/36, 1/36]
apri_prob = [round(i, 4) for i in apri_prob]
print(apri_prob, len(apri_prob))
```

課題 2.13

```
def countup(n, i):
  if i <=  n:
    print(i, end = ' ')
    countup(n, i+1)
  else:
   return

n = int(input('Enter a number: '))
countup(n, 0)
```

課題 2.14

```
def fact_r(n):
    if n == 0:
        return 1
    else:
        return fact_r(n-1) * n                  # 再帰呼び出し

n = int(input('Enter an integer n: '))        # n = 100 で OK
print(fact_r(n))
import math
math.factorial(n)
```

n = 1000 では，RecursionError: maximum recursion depth exceeded in comparison が起きます。

課題 2.15

```
def C(n, r):
  if r <= 0 or n <= r:
        return 1
  else:
        return(C(n-1, r-1) + C(n-1, r))
```

```
N = int(input('Enter a max number of rows: '))
print()
for n in range(N+1):                                  # 第0行から第N行までのパスカルの三角形
  list_C = [C(n, r) for r in range(n+1)]    # 各行の nCr のリスト
  print(n, list_C)
  print('  '.join(map(str, list_C)).center(130))   # 数字の間は2個の半角の空白
```

課題 2.16

```
def Pascal_triangle(N):
  from math import factorial as fact
  def C(n, r):
    return fact(n) // (fact(r)*fact(n -r))
  for n in range(N+1):
    list_C = [C(n, r) for r in range(n+1)]
  return list_C
sum2 = []
N = int(input('Enter a max number of rows = '))
for i in range(N+1):
  list_C = Pascal_triangle(i)
  list_C2 = []
  for i in list_C:
    list_C2.append(i*i)
  sum2.append(sum(list_C2))
print(sum2)                          # [1, 2, 6, 20, 70, 252, 924, . . . ]
```

def Pascal_triangle(N)の内部で，def C(n,r) が定義されていることに注意。

課題 2.17

```
def fizzbuzz(num):
  if num % 3 == 0 and num % 5 == 0:
    return 'Fizz Buzz'
  elif num % 3 == 0:
    return 'Fizz'
  elif num % 5 == 0:
    return 'Buzz'
  else:
   return num

n = int(input('Enter an integer n = '))
listx = list(range(1, n+1))
print(list(map(fizzbuzz, listx)))   # [1, 2, 'Fizz', 4, 'Buzz', 'Fizz', , , , ]
```

課題 2.18

```
def fibo(n):
  x, y = 0, 1
  for i in range(n):
    x, y = y, x + y
  return y

n = int(input('Enter an integer n:  '))      # F1, . . ., Fn。第 n 項まで求める
list_fibo = [fibo(i) for i in range(n)]      # [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
print(list_fibo, len(list_fibo))              #  F1 F2                          F10
```

別解

```
def fibo(n):                                 # 再帰関数の定義。メモ化しない場合
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else:
    return fibo(n-1) + fibo(n-2)

n = int(input('Enter an integer n = '))     # n = 10
list_fibo = [fibo(i) for i in range(n+1)]
del list_fibo[0]                             # F0 を削除。F1, F2, . . . , F10
print(list_fibo, len(list_fibo))            # [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

再帰を使うプログラムでは，メモ化しないと n が大きくなるときわめて非効率となります。メモ化については，課題 5.18 を参照。メモ化で再帰する場合については，前著 p157 例題 6.14 を参照してください。

課題 2.19

```
def fibo(n):
  x, y = 0, 1
  for i in range(n):
    x, y = y, x + y
  return y

n = int(input('Enter an integer n: '))
list_fibo = [fibo(i) for i in range(n)]
print(list_fibo, len(list_fibo))
list_fibo_sq = [i*i for i in list_fibo]
if sum(list_fibo_sq) == fibo(n-1) * fibo(n):
```

```
  print(True, ': F1*F1 + F2*2 + … + Fn*Fn = Fn * Fn+1 = ', fibo(n-1) * fibo(n))
else:
  print(False)
```

課題 2.20

プログラム 1

```
def palindrome(word):
  i = 0
  j = len(word) - 1
  while i < j:
    if word[i] != word[j]:
      return False                              # False の判定
    i += 1
    j -= 1
  return True                                    # True の判定
word = input('Enter a string or a digit: ')
print(palindrome(word))
```

プログラム 2

```
import palindrome
word = '12345554321'                             # 数字列の例
print(palindrome.palindrome(word))             #  True, 回文数

# モジュールファイル  palindrome.py
def palindrome(word):
  if len(word) <= 1:
    return True
  if word[0] != word[-1]:
    return False
  w = word.replace(word[0], '')
  w = word.replace(word[-1], '')
  return palindrome(w)
```

課題 2.21

```
def palindrome(word):
  i = 0
  j = len(word) - 1
  while i < j:
    if word[i] != word[j]:
      return False
    i += 1
```

```
    j -= 1
  return True
words = ['akasaka', 'asakusa', 'ABBA', '1234554321', '12354321']
list_pal = []
for i in filter(palindrome, words):
  list_pal.append(i)
print(list_pal)                  # ['akasaka', 'ABBA', '1234554321']
```

課題 2.22＊

```
def get_checkdigit(ISBN12):                # 12 桁の ISBN の数字列。checkdigit を求める
  sum_oddd, sum_evend = 0, 0
  for i, elm in enumerate(ISBN12, 1):     # index i は 1 から始まる
    if i % 2 != 0:
      sum_oddd += int(elm)                  # 奇数桁の和
    else:
      sum_evend += int(elm)                 # 偶数桁の和)
  num = sum_oddd + sum_evend * 3
  r = num % 10                             # 下 1 桁の数を取り出す
  if r == 0 :                              # 下 1 桁の数が zero
    checkdigit = str(r)                   #  checkdigit の数字
  else :                                    # 下 1 桁の数が non-zero
    checkdigit = str(10 - r)               # checkdigit の数字
  return checkdigit                        # valid checkdigit を返す

code = input('Enter a 13-bit ISBN with hyphens: ') # ハイフン付きの 13 桁の数字列
ISBN13 = code.replace('-', '')             # ハイフンを除去した 13 桁の数字
last_digit = ISBN13[len(ISBN13)-1]        # 13 桁の最後の数字。checkdigit?
ISBN12 = ISBN13[:-1]                        # 12 桁までの数字
checkdigit = get_checkdigit(ISBN12)       # valid checkdigit を返す
if checkdigit == last_digit:               # last-digit は valid checkdigit
  print('the 13-digit ISBN is valid.')
if checkdigit != last_digit:               # last_digit は not valid
  print('the 13-digit ISBN is not valid.')
  print('valid checkdigit', checkdigit)
  print('the valid 13-digit ISBN:', 'ISBN' + code[:len(code)-1] + checkdigit)
```

実行結果の一例

```
Enter a 13-bit ISBN with hyphens: 978-4-7649-6050-8
the 13-digit ISBN is not valid.
valid checkdigit 3
the valid 13-digit ISBN: ISBN978-4-7649-6050-3
```

課題 2.23

```
def split_string_func(str_seq, N, n):
  split_slist = [str_seq[i:i+n] for i in range(0, N, n)]
  split_string = ' '.join(split_slist)
  return split_string

import random
import string
#random.seed(110)
N = int(input('Enter a length of a random list: '))      # N = 16
n = int(input('Enter a length of a split_digit: '))      # n = 4
rlist = [random.choice(string.digits) for i in range(0, N)]
print(rlist, len(rlist))
code = ''.join(rlist)
print('a 16_digit code without spaces:', code)
split_code = split_string_func(code, N, n)
print('a 16_digit code with a space for every four digits:', split_code)
```

課題 2.24＊

モジュールファイル　get_ckdigit_CCN.py

```
def get_ckdigit_CCN(figs, first_digit):   # figs は，逆順の 15 桁の数字列
  sum_odd, sum_even = 0, 0
  for i, j in enumerate(figs, 2):     # i は 2 から始まる。j はカード番号の数字
    if i % 2 != 0:                      # インデックスが奇数
      sum_odd += int(j)                 # 数字を数に変換
    else:                                # インデックスが偶数
      n = int(j) * 2                    # 偶数インデックスの数字は 2 倍する
      if n >= 10:
        sum_even += n - 9
      else:
        sum_even += n
  sum_ccn = sum_odd + sum_even      # sum_odd には，インデックス 1 の数は含まれていない
  list_sum = [sum_ccn, sum_odd, sum_even]
# 有効な checkdigit を求める。余りは sum_ccn % 10
  if sum_ccn % 10 == 0:             # 15 桁で余りが 0 ならば
    checkdigit = 0                   # checkdigit は 0。これで 16 桁で 10 で割り切れる
  else:                               # 15 桁で余りが 1～9 であれば
    checkdigit = 10 - sum_ccn % 10      # これにより 16 桁で 10 で割り切れる
  checkdigit = str(checkdigit)          # 整数を数字に変換。有効な checkdigit
  if checkdigit == first_digit:         # 16 桁の番号で checkdigit を検査するとき
    TF = True                          # checkdigit は有効
  else:
```

```
    if first_digit != None:       #  first_digit として，文字列があれば
      TF = Flase          #  16 桁の番号で checkdigit を検査する。checkdigit は無効
    else:                           # first_digit == None  値がない。15 桁のカード番号
      TF = None             # 15 桁の番号に有効な checkdigit を加えるとき，None を返す。
  return TF, checkdigit, list_sum   # False でも有効な checkdigit を返す
```

関数 enumerate(イテラブル, start = 数値)では，第 2 引数に数値があればイテラブルのインデックスはその数値から始ます。16 桁のカード番号（数字列）のインデックス 1 はチェックディジットで，15 桁のカード番号はインデックスは 2 から始まることに注意してください。なお，この関数は，課題 2.25 でも使えるようなプログラムにしています。

```
#  プログラム本体
import get_ckdigit_CCN
code = input('Enter a 16-digit number: ')   # 番号には 4 桁ごとに空白を入れる
figs = code.replace(' ', '')           # 16 桁のカード番号を連続した数字列にする
figs = figs[::-1]                       # 逆順の数字列とする
print(' Reversed 16-digit:', figs, '\t', len(figs))
first_digit = figs[0]                # 16 桁のカード番号の checkdigit。これが有効か？
print('first digit:', first_digit)
figs = figs.replace(first_digit, '', 1)          # first_digit を削除した 15 桁の数字
print(' Reversed 15-digit:', figs, '\t', len(figs))
TF, checkdigit, list_sum = get_ckdigit_CCN.get_ckdigit_CCN(figs, first_digit)
list_sum[1] = int(checkdigit) + list_sum[1]      # 16 桁での sum_odd
list_sum[0] = int(checkdigit) + list_sum[0]      # 16 桁での sum_ccn
print('TorF:', TF, '  valid checkdigit:', checkdigit,\
      '  sum_ccn, sum_odd, sum_even:', list_sum)
print('the valid 16-digit credit card number:', code[:len(code)-1] + checkdigit)
```

16 桁のカード番号を逆順にしていますので，先頭の数字がチェックディジットです。カードでのインデックス番号 1 がチェックディジットで，これの有効／無効が分からないので，これをとりあえず first digit としています。チェックディジットを削除して 15 桁の数字列 figs としますと，15 桁の先頭数字のカード番号でのインデックスは 2 となります。

最後の行では checkdigit の True/False に関わらず，16 桁の正しいカード番号を出力しています。

プログラム 2： 16 桁のカード番号が有効か／無効かを判定するだけのプログラムは，簡単です。

```
def ckdigit_CCN(figs):                  # figs は，逆順の 16 桁の番号
  sum_odd, sum_even = 0, 0
  for i, j in enumerate(figs, 1):     # i は 1 から始まる。j はカード番号の数字
    if i % 2 != 0:                       # インデックスが奇数
      sum_odd += int(j)                  # 数字を数に変換
    else:                                 # インデックスが偶数
      n = int(j) * 2
      if n >= 10:
        sum_even += n - 9
      else:
```

```
        sum_even += n
  sum_ccn = sum_odd + sum_even
  if sum_ccn % 10 == 0:
    return True
  else:
    return False

code = input('Enter a 16-digit credit number: ')   # 番号には 4 桁ごとに空白を入れる
figs = code.replace(' ', '')           # 16 桁のカード番号を連続した数字列にする
figs = figs[::-1]                       # 逆順の数字列として番号を読む
print(ckdigit_CCN(figs))                # True or False の出力
```

課題 2.25＊

```
import get_ckdigit_CCN
code = input('Enter a 15-digit number: ')   # 番号には 4 桁ごとに空白を入れること
figs = code.replace(' ', '')                 # 番号を連続した数字列にする
figs = figs[::-1]                            # 逆順の数字列とする
first_digit = None              # 15 桁なので，16 桁の first_digit は None とする。
print(figs, '\t', len(figs))
TF, checkdigit, list_sum = get_ckdigit_CCN.get_ckdigit_CCN(figs, first_digit)
list_sum[1] = int(checkdigit) + list_sum[1]    # 16 桁での sum_odd
list_sum[0] = int(checkdigit) + list_sum[0]    # 16 桁での sum_ccn
print('TorF:', TF, '  valid checkdigit:', checkdigit,\
      '  sum_ccn, sum_odd, sum_even:', list_sum)
print('the valid 16-digit credit card number:', code[:len(code)] + checkdigit)
```

2.3 Mtplotlib を使ってグラフを描く

課題 2.26

```
import matplotlib.pyplot as plt
def fibo(n):
  x, y = 0, 1
  for i in range(n):
    x, y = y, x + y
  return y

n = int(input('Enter an integer n: '))
list_fibo = [fibo(i) for i in range(n)]
list_fibo_gr = []
for i in range(n-1):
  list_fibo_gr.append(list_fibo[i+1]/list_fibo[i])
```

```
plt.plot(list_fibo_gr, marker = 'o')
plt.xlim(0, n-2)
plt.ylim(1.0, 2.1)
plt.show()
print('フィボナッチ数列', list_fibo, len(list_fibo))
print('黄金比に漸近する', list_fibo_gr, len(list_fibo_gr))
print('漸近値：', list_fibo_gr[18])                    # 1.6180339631667064
print('黄金比：', (1 + 5**0.5)/2)                      # 1.618033988749895
```

課題 2.27

プログラム 1

```
import matplotlib.pyplot as plt
def fibo(n):
  x, y = 0, 1
  for i in range(n):
    x, y = y, x + y
  return y

n = int(input('Enter an integer n: '))
list_fibo = [fibo(i) for i in range(n)]          # フィボナッチ数列
list_fibo_rv = [1/fibo(i) for i in range(n)]   # フィボナッチ数の逆数列
list_fibo_rv_rt = []
for i in range(n-1):
  list_fibo_rv_rt.append(list_fibo_rv[i+1]/list_fibo_rv[i])
plt.plot(list_fibo_rv_rt, marker = 'o')
plt.xlim(0, n-2)
plt.ylim(0, 1.1)
plt.show()
#print('フィボナッチ数列', list_fibo)
#print('フィボナッチ数の逆数列', list_fibo_rv)
print('逆数の漸近値：', list_fibo_rv_rt[n-2])          # 0.6180339985218034
print('黄金比の逆数：', 1 / ((1 + 5**0.5)/2))          # 0.6180339887498948
print('フィボナッチ数列の逆数和:', sum(list_fibo_rv))   # 3.3596464890102373
```

プログラム 2

```
import matplotlib.pyplot as plt
def fibo(n):
  x, y = 0, 1
  for i in range(n):
    x, y = y, x + y
  return y

n = int(input('Enter an integer n: '))
```

```
list_fibo = [fibo(i) for i in range(n)]        # フィボナッチ数列
list_fibo_rv = [1/fibo(i) for i in range(n)]   # フィボナッチ数の逆数列
listx = []
list_sum = []
for i in list_fibo_rv:
  listx.append(i)
  list_sum.append(sum(listx))
x = np.arange(0, n, 1)
y = np.array(list_sum)
plt.plot(x, y, marker = 'o')
plt.xlim(0, n)
plt.ylim(0, 4)
plt.show()
```

課題 2.28＊

```
import numpy as np
import matplotlib.pyplot as plt
s, n = 0, 1000
list_sum = []
for i in range(1, n+1):
  s += 1/i**2
  list_sum.append(s)
x = np.arange(1, n+1, 1)
y = np.array(list_sum)
plt.plot(x, y)
plt.show()
print(s)
print(np.pi**2/6)     # pi*pi/6 = 1 + 1/2^2 + 1/3^2 + 1/4^2 + . . .
```

収束は遅い。

課題 2.29

解答省略（前著 p.189 例題 7.14 を参照）

課題 2.30

```
import numpy as np
import matplotlib.pyplot as plt
def test_stat(x):
  meanx = np.mean(x)
  medianx = np.median(x)
```

```
  stdx = np.std(x, ddof= 1)
  maxx = max(x)
  q75 = np.percentile(x, 75)
  q50 = np.percentile(x, 50)
  q25 = np.percentile(x, 25)
  iqr = q75 - q25
  minx = min(x)
  return meanx, medianx, stdx, maxx, q75, q50, q25, iqr, minx
x = [83, 79.5, 50, 68, 72, 60.5, 90, 85, 55, 95]
print(len(x))
meanx, medianx, stdx, maxx, q75, q50, q25, iqr, minx = test_stat(x)
print('平均値 ', round(meanx, 1))
print('中央値', round(medianx, 1))
print('標本標準偏差', round(stdx, 2))
print('降順ソート', sorted(x, reverse = True))
print('最大値', maxx)
print('第 3 四分位数', round(q75, 1))
print('中央値（第 2 四分位数）', round(q50, 1))
print('第 1 四分位数', round(q25, 1))
print('四分位範囲', round(iqr, 1))
print('最小値', minx)
fig, ax = plt.subplots()
labels = ['x']
ax.boxplot(x, labels = labels)
plt.show()
```

課題 2.31＊

プログラム 1：正規分布を描く

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
x = np.linspace(0, 120, 200)
y = norm.pdf(x, 62.5, 10.64)
plt.plot(x, y)
plt.show()
```

プログラム 2：サンプリング，平均値，中央値，標準偏差

```
import numpy as np
np.random.seed(100)
n = 20
data_w = np.round(np.random.normal(62.5, 10.64, n), 1)
print('sample:', n, data_w)
sumw = 0
for x in data_w:
```

```
  sumw += x
meanw = sumw / n
print('mean:', round(meanw, 1))                          # 平均
print('mean:', round(np.mean(data_w), 1))
data_sort = np.sort(data_w)
print('sorted data:', data_sort)
if n % 2  != 0:
  medianw = data_sort[int((n-1)/2)]
  print(int((n-1)/2))
else:
  medianw = (data_sort[int(n/2)-1] + data_sort[int(n/2)]) / 2
  print(int(n/2)-1, int(n/2))
  print(data_sort[int(n/2)-1], data_sort[int(n/2)])
print('median:', round(medianw, 1))
print('median:', round(np.median(data_sort), 1))
dsumw2 = 0
for x in data_w:
  dev = x - meanw
  dsumw2 += dev * dev
varw = dsumw2 / (n - 1)
stdw = varw ** 0.5                                       # 標準偏差
print('sample std:', round( stdw, 2))
print('sample std:', round(np.std(data_w, ddof=1), 2))
```

要約統計量を求めるには普段は関数を使うにしても，定義式に従うプログラムの作成を一度はやっておくとよい。ただし，中央値を求めるときには Numpy モジュールの np.sort(配列) 関数を使ってソートされた配列を用いてかまわない。中央値の計算では，データサイズが奇数か偶数で処理を分け，配列（リスト）の index は 0 から始まることに注意すること。

プログラム３：箱ひげ図を描く

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(100)
data_w = np.round(np.random.normal(62.5, 10.64, 20), 1)
x = np.round(data_w, 1)
fig, ax = plt.subplots()
labels = ['x']
ax.boxplot(x, labels = labels)
plt.show()
q75 = np.percentile(x, 75)
q50 = np.percentile(x, 50)
q25 = np.percentile(x, 25)
iqr = q75 - q25
#print('降順ソート', sorted(x, reverse = True) )
print('最大値', max(x), '    第 3 四分位数', round(q75, 1))
```

```
print('中央値（第 2 四分位数）', round(q50, 1),
      '    第 1 四分位数', round(q25, 1))
print('四分位範囲', round(iqr, 1), '    最小値', min(x))
```

課題 2.32＊

```
import numpy as np
import matplotlib.pyplot as plt
def area(a, b):
  h = 0.1
  return 0.5*(a + b)* h

x = np.arange(0, 1.1, 0.1)
y = np.array([0.0, 0.0343, 0.0822, 0.1393, 0.2070,
              0.2872, 0.3810, 0.4899, 0.6173, 0.7710, 1.0])   # ローレンツ曲線
y1 = np.arange(0, 1.1, 0.1)                                    # y = x
plt.plot(x, y, marker = 'o')
plt.plot(x, y1, marker = 's')
plt.xticks(np.arange(0, 1.1, step = 0.1))           # 目盛間隔をを詳しく設定する
plt.yticks(np.arange(0, 1.1, step = 0.1))
plt.xlim(0, 1.0)                                      # x 軸と y 軸の始点を揃える
plt.ylim(0, 1.0)
plt.grid()
plt.show()

s = 0.
for i in range(0, 10):
  s += area(y[i], y[i+1])                             # ローレンツ曲線と x 軸が囲む面積
gini = (0.5 - s) * 2
# y = x と x 軸が囲む面積(0.5)
print('\n', 'Gini coefficient:', round(gini, 4))     # ジニ係数 0.2982
```

第 3 章　古代ギリシャの三賢人のアルゴリズム

3.2　最大公約数（GCD）を求めるユークリッドの互除法

課題 3.1

```
def gcd(x, y):
  while True:
    if x < y:
      x, y = y, x
```

```
    d = x - y
    if d == x:
      return x
    print(x, y, d, sep = '\t')
    x, y = y, d
    if d == 0:
      return x

x, y = map(int, input('Enter two digits: ').split())   # 2 整数の入力
print('最大公約数 GCD: ', gcd(x, y))
```

課題 3.2

```
def find_divisors(n):              # すべての約数を求める関数
  list_divisors = []
  for i in range(1, n+1) :         # 1, 2,  . . . , n, 1 and n は必ず約数になる
    if n % i == 0:                  # 余りがゼロならば
      list_divisors.append(i)
  return list_divisors

n1 = int(input('Enter an integer = '))
n2 = int(input('Enter an integer = '))
list_divisors_n1 = find_divisors(n1)                    # n1 についてのすべての約数
print(list_divisors_n1, len(list_divisors_n1))
list_divisors_n2 = find_divisors(n2)                    # n2 についてのすべての約数
print(list_divisors_n2, len(list_divisors_n2))
cd = set(list_divisors_n1) & set(list_divisors_n2)    # 集合の AND 演算（共通集合）
print('common divisors:', cd)                            # 公約数（共通する約数）
gcd = max(cd)                                              #  最大公約数
```

課題 3.3

```
import sympy
x, y = map(int, input('Enter two digits: ').split())
print('sympy による x, y の最大公約数:', sympy.igcd(x, y))   # ユークリッドの互除法
if sympy.igcd(x, y) == 1:
  print('最大公約数が 1 で，互い素です')
else:
  print('互いに素ではありません')
print()
print('sympy による x の素因数分解', sympy.factorint(x))
print('sympy による y の素因数分解', sympy.factorint(y))
listx= [ k for k in sympy.factorint(x).keys()]
listy= [ k for k in sympy.factorint(y).keys()]
```

```
comfact = set(listx) & set(listy)                    # 共通集合を求める
print('素因数の共通集合', comfact)
if len(comfact) == 0:                                  # 空集合ならば
  print('共通の素因数はありません。互い素で，最大公約数は 1 です')
else:
  print('互いに素ではありません')
```

課題 3.4

プログラム 1

```
%%timeit                        # ユークリッドの互除法
def gcd(x, y):
  while y != 0:
    x, y = y, x % y
  else:
    return x
x, y = 99999, 100000
#print('最大公約数 GCD', gcd(x, y))
```

プログラム 2

```
%%timeit                              # 素因数分解による方法
def try_division(num) :             # １つの自然数に対して，素因数分解を辞書で表す
    list_pf= []
    divisor = 2                      # # 除数。２は最初の素数
    while num >= 2:          # 繰り返し。  num = 1 になったら while ループを抜ける
      q, r = divmod(num, divisor)
      if r == 0:                       # 余り 0。割り切れる
        list_pf.append(divisor)      # 約数としてリストに収める
        num = q                   # 商を新たな num にして，さらに割り切れるかを続ける
      else:
        divisor = divisor + 1       # 割り切れないときには +1 して，次の除数に移る
    dict_pf = {i: list_pf.count(i) for i in list_pf}    # 辞書に変換
    return dict_pf

x, y = 99999, 100000
divisors1 = try_division(x)               # x の素因数分解の辞書。
#print('x の素因数分解の辞書：', divisors1)
divisors2 = try_division(y)               # y の素因数分解の辞書
#print('y の素因数分解の辞書：', divisors2)
listc = list(set(divisors1.keys()) & set(divisors2.keys()))
#print('共通のキー（素因数）のリスト：', listc)       # 共通のキー（素数）のリストを得る
gcd = 1
for i in range(0, len(listc)):
```

```
  if divisors1[listc[i]] <= divisors2[listc[i]]:   # 値の小さい方を選ぶ
    gcd *= listc[i] ** divisors1[listc[i]]          # 1 つの素数のべき乗の累積
  else:
    gcd *= listc[i] ** divisors2[listc[i]]
#print('最大公約数：', gcd)
```

最大公約数 1 を求める素因数分解による方法は，ユークリッドの互除法と比べて，おおよそ 1000 倍の実行時間を要すること，言い換えれば，ユークリッドの互除法は，素因数分解による方法と比べてはるかに高速ということが分かります。

課題 3.5

```
import sympy
n = int(input('Enter an integer n for Fn : '))
list_fibo = [sympy.fibonacci(i) for i in range(1, n+1)]
size = len(list_fibo)
print(list_fibo, size)
for i in range(size):
  if i == size - 1:
    break
  else:
    num1, num2 = list_fib[i], list_fib[i+1]
    x = sympy.gcd(num1, num2)       # GCD を求める
    if x == 1:
      print(num1, num2, '  GCD 1, 互いに素です')
    else:
      print(num1, num2, '  互いに素ではない')
```

課題 3.6

```
def try_division(num) :              # 素因数分解を辞書形式でで求める関数の定義
    list_pf= []
    divisor = 2                       # 除数。2 は最初の素数
    while num >= 2:             # 繰り返し。num = 1 になったら while ループを抜ける
      q, r = divmod(num, divisor)
      if r == 0:                      # 余り 0。割り切れる
        list_pf.append(divisor)     # 約数としてリストに収める
        num = q                  # 商を新たな num にして，さらに割り切れるかを試みる
      else:
        divisor = divisor + 1      # 割り切れないときには +1 して，次の除数に移る
    dict_pf = {i: list_pf.count(i) for i in list_pf}    # 辞書に変換
    return dict_pf

n = int(input('Enter an integer n: '))
```

```
dict_pf = try_division(n)
print(dict_pf)
if 2 in dict_pf and len(dict_pf) == 1:      # 2 がキー　かつ　辞書のサイズが 1 ならば
  print(n, 'は 2 の冪乗で表される')            # 辞書 {2: 冪指数}
else:
  print(n, 'は 2 の冪乗で表されない')
```

3.3 円周率を求めるアルキメデスの方法

課題 3.7

```
import numpy as np
def archi_pir(x, y, i, m):
  if i == m:
    return (x + y) / 2
  x = (2*x*y) / (x + y)
  y = np.sqrt(x*y)
  print('第', i+1, '近似　正', 2**(i+1)*n, '角形　', y, '\t', x)
  return archi_pir(x, y, i+1, m)

n = 6              # 正 6 角形からスタート，第 0 近似から第 m 近似まで
m = int(input('Enter the max order of approximation: '))
x, y = 2*np.sqrt(3), 3
print('第 0 近似　正', n, ' 角形　', y, '\t', x)
i = 0
ap_pi = archi_pir(x, y, i, m)
print('円周率の近似値：', round(ap_pi, 6))
```

課題 3.8

```
sumx = 1
for i in range(0, 50000, 2):                # 終了値を巨大にしても，精度はよくない。
  sumx *= (i+2)*(i+2)/((i+1) * (i+3))
print('π の近似値', 2 * sumx)                 # 3.141435593589904
```

課題 3.9

```
from fractions import Fraction
N = 200
M = 200
list_t1 = [Fraction((-1) ** n, (2 * n + 1) * 5 ** (2 * n + 1)) for n in range(N)]
list_t2 = [Fraction((-1) ** n, (2 * n + 1) * 239 ** (2 * n + 1)) for n in range(M)]
```

```
sum1 = sum(list_t1)
sum2 = sum(list_t2)
print('π の近似値：', 16 * float(sum1) - 4 * float(sum2))
```

課題 3.10

プログラム(1)

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 1)
y = np.sqrt(1.0 - x*x)                                  # 単位円
の方程式　x*x + y*y = 1
plt.plot(x, y)
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.show()
```

プログラム(2)

```
import numpy as np
def f(x):
  return np.sqrt(1.0 - x*x)

a, b = 0, 1                                               # 下限値，上限値
n = int(input('Enter a number of divisions '))   # 分割数
h = (b - a) / n
s = 0
for i in range(n):
  xi = a + i * h
  s += 0.5 * (f(xi) + f(xi + h)) * h              # 台形公式による区分積分値（面積）s
print('n =', n, 'での π の近似値: ', 4*s)          #  π = 4*s
print('Numpy の定数 π: ', np.pi)


import numpy as np
from scipy import integrate
from scipy import constants
def f(x):
  return np.sqrt(1 - x*x)

a, b = 0, 1
s, err = integrate.quad(f, 0, 1)                    # 関数と積分区間を与えて定積分
print('Scipy 定積分を用いて求めた π の近似値 ', 4*s)
print('scipy の π ', constants.pi)
```

課題 3.11

```
def mnemonic(phrase):
  wchs = ''
  for i, word in enumerate(phrase, 1):
    wchs += str(len(word))
    if i == 1:
      wchs += '.'
  return wchs

phrase = 'May I have a large container of coffee'
phrase = phrase.split()                 # 単語を要素とするリスト。
print(phrase, len(phrase))
print(float(mnemonic(phrase)))          # float 型に変換。3.1415926
```

同様にしてネイピア数の値を得ることができます。

3.4 素数列を求めるエラトステネスの篩の方法

課題 3.12

```
import sympy
n = int(input('Enter an integer n for Fn : '))
list_fibo = [sympy.fibonacci(i) for i in range(1, n+1)]
print(list_fibo, len(list_fibo))
print('i', 'Fi', sep = '\t')
for i in range(2, n+2, 2):              # 2, 4, 6, 8,  even number
  fi = sympy.fibonacci(i)
  if sympy.isprime(fi):
    print(i, fi, '素数', sep = '\t')
  else:
    if i == 2:
      print(i, fi, '素数でも合成数でもない',  sep = '\t')
    elif i != 2:                         # 4, 6, 8,
      print(i, fi, '合成数',  sep = '\t')
```

課題 3.13

```
import pprint
def primes(n, list_prime, i):
  if i <= n**0.5:
    for j in range(i*i, n+1, i):
```

```
      list_prime[j] = False
    i += 1
    return primes(n, list_prime, i)                ## 再帰呼び出し。
  else:
    list_prime = [ k for k in range(n + 1) if list_prime[k]]
    return list_prime, len(list_prime)

n = int(input('Enter an integer n = '))
list_prime = [True] * (n + 1)
list_prime[0] = False
list_prime[1] = False
i = 2                 # 初期化，最初の素数は２
list_prime, size = primes(n, list_prime, i)      # 関数の呼び出し
print(n, 'までのすべての素数と素数の個数')
pprint.pprint(list_prime, width = 60, compact = True)
print(size)
```

課題 3.14

```
def prime_number(n, i=2):
  if n == i:
    return True
  elif n % i == 0:
    return False
  return prime_number(n, i + 1)

n = int(input('Enter an integer n: '))
list_prime = [i for i in range(2, n) if prime_number(i) ]
print(list_prime, len(list_prime))
```

課題 3.15

解答省略

第 4 章　『博士の愛した数式』のプログラミング

4.1　素数の不思議

課題 4.1

```
import sympy
primes = list(sympy.primerange(3, 100))
print(primes)
print('素数の総数', len(primes))
listx = []
listy = []
for n in range(1, 25):       # 1, 2, ..., 24 個の素数を分類する
  x = 4*n + 1
  if x in primes:
    listx.append(x)
  y = 4*n -1
  if y in primes:
    listy.append(y)
print('4n + 1 に分類', listx, ' 個数', len(listx))
print('4n - 1 に分類', listy, ' 個数', len(listy))
print('素数の個数の和', len(listx) + len(listy))
```

課題 4.2

```
import numpy as np
import matplotlib.pyplot as plt
import sympy

x = np.array([10, 10**2, 10**3, 10**4, 10**5,
              10**6, 10**7, 10**8, 10**9, 10**10])      # 整数
y = []
for i in range(1, 11) :         # 10, 10**2, 10**3, . . .,  10**10(100 億)
  n = 10 ** i
  y.append(sympy.primepi(n))
y = np.array(y)
print('π(n)', y, len(y))

yg = []
for i in range(1, 11) :       # 10, 10**2, 10**3, . . . ,  10**10(100 億)
  n = 10 ** i
  yg.append(int(n / np.log(n)))             # 自然対数
yg = np.array(yg)                            # 推測値
print('漸近値', yg, len(yg))

plt.scatter(x, y, marker = 'o', s = 30)     #  真値
plt.scatter(x, yg, marker = 'x', s = 30)    # ガウス推測値
plt.xscale('log')
plt.yscale('log')
plt.xlabel('integer')
plt.ylabel('number of prims')
```

```
plt.show()

print()
ratio = []
for i in range(0, 9):
  ratio.append(round(y[i]/yg[i], 3))
print(ratio)
```

課題 4.3

```
import itertools
import sympy
digit = '123456789'
i = 0
for v in itertools.permutations(digit):
  i += 1
  v = int(''.join(v))
  if sympy.isprime(v):
    print(v, True)
print('順列の総数', i,  '  no primes')
```

課題 4.4

```
import sympy
num = 31
list_num =[i for i in str(num)]
print(list_num)
print(num)
for i in range(1, 10):
  if sympy.isprime(num):
    print(num, True)
  else:
    print(num, False)
  if i <= 8:
    list_num.insert(0, '3')
    print(list_num, len(list_num))
    num = int(''.join(list_num))
    print(num)
```

課題 4.5

```
from sympy import isprime
n = 73939133
```

```
while n > 0:
  print(n, isprime(n))
  n = n // 10
```

課題 4.6

```
from sympy import isprime
n = 357686312646216567629137
while n > 0:
  print(n, isprime(n))
  list_digit = [i for i in str(n)]
  list_digit.pop(0)
  if list_digit == []:
    break
  n = int(''.join(list_digit))
```

課題 4.7

```
def find_list_divs(n):          # 自分自身を除く約数を要素とするリストを求める関数
  list_divs = [1]                # 1 はすべての数の約数
  for i in range(2, n):         # 2, 3, . . ., n-1。 n は含まない。
    if n % i == 0:
      list_divs.append(i)
  return list_divs               # 自分自身を除く約数のリスト

n = int(input('Enter an integer n: '))
list_divs = find_list_divs(n)
if n == sum(list_divs):
  print('True. a perfect number.', sum(list_divs))
  print(list_divs)
else:
  print('False. not a perfect number.', sum(list_divs))
  print(list_divs)
```

4.2 その他の数の不思議

課題 4.8

```
pnum = [6, 28, 496, 8128, 33550336]              # 完全数の最初の 5 個
for i, elm in enumerate(pnum):
  x = []
  for j in range(1, elm):
```

```
    if elm % j == 0:
      x.append(j)                          # 約数を要素とするリスト
  print(pnum[i], x, sum(x))
```

課題 4.9

```
list_pnum = [6, 28, 496, 8128, 33550336]       #  完全数のリスト
list_last_elm = []
k = 0
for i in list_pnum:                                  # 完全数を取り出す
  sumx = 1
  for j in range(2, list_pnum[k]+ 1):
    sumx += j                                        # 自然数の和
    if sumx == i:                                    # 連続した自然数の総和 == 完全数
      print('a perfect num:', list_pnum[k],
            '  sum of elemnts:', sumx, '  last element:', j)
      list_last_elm.append(j)
      break
  k += 1
print('list_last_elm:', list_last_elm)    # [3, 7, 31, 127, 8191]
```

課題 4.10

```
def try_division(num) :              # 素因数分解を辞書形式でで求める関数の定義
    list_pf= []
    divisor = 2                       # 除数。2 は最初の素数
    while num >= 2:           # 繰り返し。num = 1 になったら while ループを抜ける
      q, r = divmod(num, divisor)
      if r == 0:                      # 余り 0。割り切れる
        list_pf.append(divisor)      # 約数としてリストに収める
        num = q                  # 商を新たな num にして，さらに割り切れるかを試みる
      else:
        divisor = divisor + 1      # 割り切れないときには +1 して，次の除数に移る
    dict_pf = {i: list_pf.count(i) for i in list_pf}    # 辞書に変換
    return dict_pf

from sympy import isprime
list_last_elm = [3, 7, 31, 127, 8191]
for i in list_last_elm:
  dict_pf = try_division(i+1)               # i+1 の素因数分解の辞書が返される
  if 2 in dict_pf and len(dict_pf) == 1:    # {2: べき指数} であれば，i+1 は 2 の冪
    print(i, '+ 1:', dict_pf, '\t', i, 'は 2 の冪乗 − 1')  # i はメルセンヌ数
    if isprime(i):                          # i の素数判定
```

```
      print(i, ' メルセンヌ数でかつ素数')
      print()
```

課題 4.11

```
from sympy import isprime
for n in range(1, 20):
  num = 2**n - 1              # メルセンス数
  if isprime(num):            # num が素数ならば，メルセンヌ素数
    pnum = num * 2**(n-1)     # メルセンヌ素数 * 2**(n-1) は，偶数の完全数
    print(pnum)                # 6, 28, 496, . . . , 8589869056, 137438691328
```

課題 4.12

```
import pprint
n = int(input('Enter an integer n in (2^n - 1) : '))
list_mer = list([2**i - 1 for i in range(1, n+1)]  )     # メルセンヌ数
list_merb = []
for i in list_mer:
  list_merb.append(bin(i))   # bin()は引数に渡した整数を 2 進数表現の文字列に変換
print('メルセンヌ数', list_mer)
print('メルセンヌ数の 2 進数表記', list_merb)
```

課題 4.13

```
def amifunc(pair):
  for n in pair:           # pair がタプル
    x = [i for i in range(1, n) if n % i == 0]     # 約数から自分自身を除く
    print('自然数', n, '  約数のリスト', x, '  約数の和', sum(x))

anum = [(220, 284), (1184, 1210), (2620, 2924),
        (5020, 5564), (6232, 6368)]                  # 要素は友愛数のペア(タプル)
for i in anum:
  amifunc(i)
  print()
```

課題 4.14

```
from sympy import factorint
def factorization(n):
  dictn = factorint(n)                 # 素因数分解，辞書形式　素因数：冪乗
  sumn = 0
```

```
  for k, v in dictn.items():
    sumn += k * v                          # キー * 値，素因数の和，   k**v ではない。
  return n, dictn, sumn                    # 戻り値はタプル

x = int(input('Enter a digit: '))
x, dictx, sumx = factorization(x)
print(x, dictx, sumx)                      # 自然数，素因数分解，素因数の和
y = x + 1                                  # 2 つの連続した自然数
y, dicty, sumy = factorization(y)
print(y, dicty, sumy)
if sumx == sumy:                           # 素因数の和が等しい
  print(True)
else:
  print(False)
```

課題 4.15

```
def divisors(n):
  list_divisors = []
  for i in range(1, n+1) :                 # 1, 2,  . . . , n
    if n % i == 0:                         # 余りがゼロならば
      list_divisors.append(i)
  return list_divisors, len(list_divisors)

n = int(input('Enter an integer n = '))
listx = []
for i in range(1, n+1):
  listx.append(i**2)
print(listx)
for j in listx:
  list_divisors, size = divisors(j)
  if size % 2 != 0:                        # 奇数
    print('平方数', j, ' 約数のリスト', list_divisors, ' 個数', size, ' ', True)
```

課題 4.16

```
import sympy
n = int(input('Enter an integer n = '))    # n までの素数を求める
primes = list(sympy.primerange(2, n))
m = 10                                     # 被除数
list_pm1 = []                              # 循環節の長さ p-1 をもつ素数列を収める
for p in primes:
  listqs, listrs = [], []                  # 商のリスト，余りのリスト
```

```
  q, r = divmod(m, p)          # 商，余り= divmod(被除数，除数)，除数 p 固定
  listqs.append(q)
  listrs.append(r)
  if r == 0:                     # 有限小数。分母の素数が 2，5 は，有限小数となる
    continue
  while True:
    r = r * 10
    q, r = divmod(r, p)
    listqs.append(q)
    listrs.append(r)
    if listrs[0] == r:
      break
  listqs.pop()                          # 例題 4.16  Artin.ipynb L30  変更。高速化
  len_period = len(listqs)           # 例題 4.16  Artin.ipynb L31  変更。高速化
  if len_period == p -1:             # 例題 4.16  Artin.ipynb L35  変更
    list_pm1.append(p)
print('the number of primes with period p-1:', len(list_pm1))
print('the total number of primes:', len(primes))
print('rate:', round(len(list_pm1)/len(primes), 4))
```

実行結果

```
Enter an integer n = 100000
the number of primes with period p-1: 3617
the total number of primes: 9592
rate: 0.3771
```

なお，

```
fromi sympy import sieve
n = int(input('Enter an integer n = '))
primes = list(sieve.primerange(2, n))
```

を使うほうが速くなります。

第 5 章　多様なアルゴリズムとプログラミング

5.1　英文テキストの処理

課題 5.1＊

先に，テキストファイルを作成し，ファイル名を mary_lamb.txt とします。

```
with open('merry_lamb.txt', 'w') as fobj:      #  ファイルにテキストを書き込む
  text = '''Mary had a little lamb,
```

```
Little lamb, little lamb,
Mary had a little lamb,
Its fleece was white as snow.'''   fobj.write(text)
```

テキストを「'''　'''」（引用符を 3 つ並べた対）で前後を括ると，改行を含んだ文字列とすることができます。改行した次の英文は，字下げせずに左端に寄せておくこと。字下げすると，無用な空白ができます。

```
from collections import Counter
with open('mary_lamb.txt', 'r') as fob_in:
  text = fob_in.read()
#print(text)
text2 = text.replace(',', '').replace('.', '').lower()
list_words = text2.split()           # 文字列を空白で分割した各単語を要素とするリスト
wdcounter = Counter(list_words)      # Counter オブジェクトの生成
#print(type(wdcounter))               # <class 'collections.Counter'>
#print(wdcounter.values())          # dict_values([2, 2, 2, 4, 4, 1, 1, 1, 1, 1, 1])
wdsum = sum(wdcounter.values())      # 値(出現頻度) の和を求めている。28
print('total words:', wdsum)
print('different words:', len(wdcounter))  # 重複のない異なった単語の個数
print(list_words)                      # リスト
print(wdcounter)                       # Counter オブジェクトの出力。辞書形式
```

実行結果

```
total words: 20
different words: 11
['mary', 'had', 'a', 'little', 'lamb', 'little', 'lamb', 'little', 'lamb', 'mary', 'had',
'a', 'little', 'lamb', 'its', 'fleece', 'was', 'white', 'as', 'snow']
Counter({'little': 4, 'lamb': 4, 'mary': 2, 'had': 2, 'a': 2, 'its': 1, 'fleece': 1,
'was': 1, 'white': 1, 'as': 1, 'snow': 1})
```

メソッドはドット . で連結していくことができます。

```
wdcounter = Counter(list_words)    # Counter オブジェクトの生成，引数は list_words
```

このコードでは，辞書の派生クラス（コンストラクタ）の Counte に引数としてリストを渡して呼び出して辞書型のオブジェクト wdcounter を生成（コンストラクト）しています。これは，辞書クラス（コンストラクタ）の dict にリストを渡して辞書型のオブジェクトを生成する働きと同様です。

wdcounter.values() は辞書型のビューオブジェクト（view object）といわれるもので，値を 1 つずつ返す仕組みをもつイテラブルです。イテラブルですから関数 sum(iterable）の引数になることができますので，sum(wdcounter.values()) により，値の総和を求めています。

テキストから単語を抽出する方法に，正規表現を使うことで，コードの理解は難しくなりますが，簡潔に書くことができます。正規表現の re モジュールをインポートして，次のコードを書きます。

```
list_words = re.findall(r'\b\w+\b', text.lower())
```

メソッド findall では、第 1 引数に正規表現パターンを、第 2 引数に文字列（テキスト）を指定すると，テキストの中で正規表現パターンにマッチするすべての部分文字列（今の場合は英単語）を要素とするリストを返します。ここで，英単語は英数字あるいはアンダースコアの並んだ ものとして定義されています。r'\b\w+\b' は，単語境界から始まって単語境界に終わる英単語を抽象的に表しています。ここで，第 1 引数では，raw 文字列を使用していますので，\ はエスケープ文字ではありません。

課題 5.2

```
with open('mary_lamb.txt', 'r') as fob_in:
  text = fob_in.read()
  print(text)
  text2 = text.replace('Mary', 'Betty').replace('lamb', 'goat')
  print()
  print(text2)
  print()
  print('Betty:', text2.count('Betty'))
  print('lamb:', text2.count('goat'))
  print()

with open('betty_goat.txt', 'w') as fob_out:    # ファイル名を変えて書き込む
  fob_out.write(text2)
with open('betty_goat.txt', 'r') as fob_in:
  text3 = fob_in.read()
  print(text3)
```

課題 5.3

```
import csv
import numpy as np
import matplotlib.pyplot as plt
with open('letter_freq_table.csv', 'r') as fobj:
  h = next(csv.reader(fobj))
  rdobj = csv.reader(fobj)
  list_data = [row for row in rdobj]
  print(list_data)
x, y, c= [], [], []
for i in range(len(list_data)):
  x.append(float(list_data[i-1][0]))
  c.append(str(list_data[i-1][1]))
  y.append(float(list_data[i-1][2]))
x, c, y= np.array(x), np.array(c), np.array(y)
plt.bar(x, y, tick_label = c)
plt.xlim(0, 26)
plt.ylim(0, 15)
```

```
plt.show()
```

課題 5.4

```
import csv
import pprint
import numpy as np
import matplotlib.pyplot as plt
with open('eng_freq_wiki1.csv', 'r') as fobj:
  h = next(csv.reader(fobj))                # header を読む
  robj = csv.reader(fobj)
  list_data = [row for row in robj]
#  print(list_data)                         # [['1', 'the', '56271872'], . . .  ]

list_word = []                              # 要素は単語だけのリストとする
for i in range(len(list_data)):
  list_word.append(list_data[i][1])         # 単語を appennd
#print(list_word)                           # ['the', 'of', 'and', 'to', . . . ]
#print('the size of samples:', len(list_word))

list_word_N =[[] for i in range(16)]         # 2 次元リストの初期化。A.9 節参照
for w in list_word:                          # 頻度順の単語を取り出す
    list_word_N[len(w)].append(w)            # 1 文字から 15 文字までの単語
for i in range(1, 16):
  print(i, 'letter words:', len(list_word_N[i]))
  list_word_N[i] = sorted(list_word_N[i])   # 各文字数の単語をアルファベット順にソート
  pprint.pprint(list_word_N[i], width = 60, compact = True)    # 各文字数の単語リスト
  print()
```

5..2. シーザー暗号（Caesar cipher）

課題 5.5

```
def caesar_cipher(text, k, m):
  c_text = ''
  for c in text:
    if c.isalpha():                 # アルファベットの文字ならば
      if 65 <= ord(c) <= 90:        # 大文字ならば
        n = ord(c) - 65
        n = (n + k) % 26
        n = n + 65                  # 大文字の換字
      elif 97 <= ord(c) <= 122:     # 小文字ならば
        n = ord(c) - 97
```

```
        n = (n + k) % 26
        n = n + 97                  # 小文字の換字
      c_text += chr(n)
    else:
      c_text += c                   # アルファベットの文字でない character
  return c_text

def readfile(file_name):            # ファイルを読み込む
  try:
    with open(file_name, 'r') as fob_in:
      text = fob_in.read()
  except FileNotFoundError:
    print('The file is not found')
  except Exception as err_msg:
    print('An error occurs', err_msg)
  else:
    return text

file_name = input('Enter a file name: ')
text = readfile(file_name)
print(text)
k = int(input('Enter a key number: '))
m = 26
ciphered_text = caesar_cipher(text, k, m)
print(ciphered_text)
with open(file_name, 'w') as fob_out:     # 結果のファウルへの書き込み
  fob_out.write(ciphered_text)
```

.txt のファイル名を読み込んで，キーk を与えて暗号化して，同じファイル名で書き込むようにしています。暗号化ファイルを読み込んでキーを-k にして復号化すれば，元のファイルに戻ります。

5.4 数値計算

課題 5.6

プログラム１：関数 $f(x)=\frac{1}{1+x^2}$ のグラフ

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-2, 2.1, 0.1)
y = 1 / ((1 + x*x))
plt.plot(x, y)
plt.xlim(-2, 2)
plt.ylim(0, 1.2)
```

```
plt.grid()
plt.show()
```

プログラム２

```
import numpy as np
def f(x):
  return 1 / (1 + x*x)

a, b = 0, 1                # 下限値, 上限値
n = 10000             # 分割数
h = (b - a) / n
s = 0
for i in range(n):
  xi = a + i * h
  s += 0.5 * (f(xi) + f(xi + h)) * h
print('台形公式による積分値（面積）: ', s)
print('台形公式による π の近似値 : ', 4*s)

import scipy                                    # scipy をインポート
def f(x):
  return 1 / (1 + x*x)
s, err = scipy.integrate.quad(f, 0, 1)      # 定積分の確認
print('scipy による積分値（面積）      : ', s)
print('scipy による π の近似値 1 :  ', 4*s)
print('scipy による π の近似値 2 :  ', scipy.pi)
```

課題 5.7

グラフを描くプログラム

```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return x * x - np.pi

x = np.arange(-4.0, 4.1, 0.1)
y = f(x)
plt.plot(x, y)
plt.xlim(-5, 5)
plt.ylim(-10, 20)
plt.grid()
plt.show()
```

```
# 二分法のプログラム
import numpy as np
def f(x):
  return x*x - np.pi

eps = 1.0e-8
a = float(input('Enter a: '))      # 1.5
b = float(input('Enter b: '))      # 2.5
i = 1
while(True):
  c = (a + b) / 2.
  if abs(f(c)) < eps:
    break
  elif f(c) * f(b) < 0:
    a = c
  else:
    b = c
  print('第', i, '近似値：', c)
  i += 1
  if abs(a - b)  < eps:
    break
print('近似解：', c)                              #   π の平方根
print(' numpy:', np.sqrt(np.pi))           # 確認


#  ニュートン法のプログラム
import numpy as np
def f(x):
  y = x * x - np.pi
  return y
def g(x):
  z = 2 * x
  return z

eps = 1.0e-8
x0 = 1.0
i = 1
while(True):
  x1 = x0 - f(x0)/g(x0)                  # x1 =  (x0 + np.pi/x0) / 2
  print('第', i, '近似値：', x1)
  if abs(x1 - x0) < eps:
    break
  else:
   x0 = x1
   i += 1
```

```
print('近似解：', x1)
```

実数解が 2 個あります。一般にニュートン法のほうが二分法よりも収束が速く，精度もよいことが多い。

課題 5.8

```
# グラフ
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return x**3 - 2*x**2 -x + 2

x = np.arange(-4.0, 4.1, 0.1)
y = f(x)
plt.plot(x, y)
plt.xlim(-5, 5)
plt.ylim(-10, 20)
plt.grid()
plt.show()
```

実数解が 3 個あります。1 つの解を求めるプログラムを示します。

```
# 二分法のプログラム
def f(x):
  return x**3 - 2*x**2 -x + 2

eps = 1.0e-8
a = float(input('Enter a: '))      # 1.5  下限値
b = float(input('Enter b: '))      # 2.5  上限値
i = 1
while(True):
  c = (a + b) / 2.
  if abs(f(c)) < eps:
    break
  elif f(c) * f(b) < 0:
    a = c
  else:
    b = c
  print('第', i, '近似値:', c)
  i += 1
  if abs(a - b)  < eps:
    break
print('近似解:', c)
```

```
#  ニュートン法
import numpy as np
def f(x):
  y = x**3 - 2*x**2 -x + 2
  return y
def g(x):
  z = 3*x**2 - 4* x -1
  return z
eps = 1.0e-8

x0 = 3.0                                  # 第0近似値を定める
i = 1
while(True):
  x1 = x0 - f(x0)/g(x0)
  print('第', i, '近似値', x1)
  if abs(x1 - x0) < eps:
    break
  else:
   x0 = x1
   i += 1
print('近似解', x1)
```

5.5　数式の後置記法への変換とスタック計算

課題 5.9*

(+　　②　a b +　　③　* (　　④　* (-　　⑤　a b + c d
⑤　a b + c d -　　⑦　*

課題 5.10*

infix = '((a + (b * (c - d))) / e)'		
list _infix = ['(', '(', 'a', '+', '(', 'b', '*',		
	(', 'c', '-', 'd', ')', ')', ')', '/', 'e', ')']	
rpn_queue	op_stack	操作
	((	
a		
	((+ (	
a b		
	((+ (* (	
a b c		
	((+ (* (-	
a b c d		
	((+ (* (-)	
a b c d -		
	((+ (*	
	((+ (*)	
a b c d - *		
	((+)	
a b c d - * +		
	(/	
a b c d - * + e		
	(/)	
a b c d - * + e /		

課題 5.11*

中置記法の数式で左右の括弧の対応がとれているかどうかを最初にチェックします。

。

```
import sys                                  # sys モジュールのインポート
infix = '( ( a + ( b * ( c - d ) )  + e )'      # 中置記法の括弧のある数式例
if infix.count('(')  != infix.count(')'):
   print('エラー： 括弧の左右の対応がとれていません')
   sys.exit()                          # exit() 関数を途中で終了させたい位置に置く
else:
print('左右の括弧の対応がとれています')
```

先に，プログラム本体を示します。

```
import infix_to_rpn_prt
infix = input('Enter an infix exp: ')
list_infix = infix.split()
print('list_infix', list_infix)
```

```
rpn_queue = infix_to_rpn_prt.infix_to_rpn_prt(list_infix)
print('rpn_queue ', rpn_queue)
rpn = ' '.join(rpn_queue)
print('rpn exp ', rpn)
```

以下をモジュールファイルとします。　infix_to_rpn_prt.py

```
def comp(b, a, op_stack, rpn_queue):
  if b + a in ['+*', '+/', '-*', '-/']:
    op_stack.append(b)
    op_stack.append(a)
  else:
    rpn_queue.append(b)
    op_stack.append(a)
    print('rpn_queue', rpn_queue)
    print('op_stack', op_stack)
  return op_stack, rpn_queue

def infix_to_rpn_prt(list_infix):
  rpn_queue = []
  op_stack = []
  for c in list_infix:
    if c in '()+-*/':
      if c == '(':
        op_stack.append(c)
        print('op_stack', op_stack)
        continue
      if c == ')':
        op_stack.append(c)
        print('op_stack', op_stack)
        op_stack.pop()
        rpn_queue.append(op_stack.pop())
        print('rpn_queue', rpn_queue)
        x = op_stack.pop()
        if x == '(':
          continue
        else:
          rpn_queue.append(x)
          print('rpn_queue', rpn_queue)
          if op_stack.pop() == '(':
            continue
        print('rpn_queue', rpn_queue)
      op_stack.append(c)
      print('op_stack', op_stack)
      if len(op_stack) >= 2:
        a = op_stack.pop()
```

```
        b = op_stack.pop()
        if b == '(':
          op_stack.append(b)
         op_stack.append(a)
          continue
        if b != '(':
          comp(b, a, op_stack, rpn_queue)
          if len(op_stack) == 2:
            a = op_stack.pop()
            b = op_stack.pop()
            if b == '(':
              op_stack.append(b)
              op_stack.append(a)
              continue
            comp(b, a, op_stack, rpn_queue)
    else:
      rpn_queue.append(c)
      print('rpn_queue', rpn_queue)
  print('operators left in op-stack', op_stack)
  if len(op_stack) == 2:
    rpn_queue.append(op_stack.pop())
    rpn_queue.append(op_stack.pop())
  elif len(op_stack) == 1:
    rpn_queue.append(op_stack.pop())
  return rpn_queue
```

課題 5.12

```
import dis
dis.dis('( ( a + ( b * ( c - d ) ) ) / e )')
```

実行結果

```
1           0 LOAD_NAME                0 (a)
            2 LOAD_NAME                1 (b)
            4 LOAD_NAME                2 (c)
            6 LOAD_NAME                3 (d)
            8 BINARY_SUBTRACT
           10 BINARY_MULTIPLY
           12 BINARY_ADD
           14 LOAD_NAME                4 (e)
           16 BINARY_TRUE_DIVIDE
           18 RETURN_VALUE
```

このソースコードを読むことにより，後置式 `a b c d - * + e /` を得ることができます。

課題 5.13

```
import get_rpn_queue                                    # モジュールファイルのインポート
import rpn_calc                                          # モジュールファイルのインポート
rpn = input('後置記法の数式を入力してください： ')
rpn_queue = get_rpn_queue.get_rpn_queue(rpn)
print('rpn_queue', rpn_queue)
print('計算結果 ', rpn_calc.rpn_calc(rpn_queue))
```

モジュールファイル　get_rpn_queue.py

```
def get_rpn_queue(rpn):
  list_str = rpn.split()
  print(list_str)
  rpn_queue = []
  for c in list_str:
    if c in '+-*/':
        rpn_queue.append(c)
    else:
        rpn_queue.append(eval(c))
  return rpn_queue
```

モジュールファイル　rpn_calc.py

```
def rpn_calc(rpn_queue):
  stack = []                          # 数値スタックの初期化
  for i in rpn_queue:
    if type(i) is float or type(i) is int:
      stack.append(i)                 # append は push の働き
      print('数値スタック', stack)
    else:
      print(i, '演算子が取り出されたので，二項演算を実行します')
      a = stack.pop()                 # 末尾から数値を pop して代入
      b = stack.pop()
      print('数値スタックでの数値の並び b a \n', b, a)
      print(' ', i)                   # 演算子を表示
      if i == '+':                    # 2 項演算です
        val = b + a                   # 演算では　b が先，a が後に来る
      elif i == '-':
        val = b - a
      elif i == '*':
        val = b * a
      elif i == '/':
        val = b / a
      stack.append(val)               # 2 項演算の結果を push
      print('数値スタック', stack)
```

```
  return stack.pop()              # 結果を pop
```

課題 5.14

後置記法の数式への変換は，課題 5.12 のプログラムで中置記法の数式を入力して実行します。スタック計算は，課題 5.14 のプログラムで後置記法の数式を入力して実行するか，dis モジュールの dis() 関数を使って求めます。

```
import dis
dis.dis('( ( 1 + ( 2 * ( 3 - 4 ) ) ) / 5 )')

import dis
dis.dis('5 * ( ( ( 7 + 9 ) * ( 2 * 5 ) ) + 6 )')

import dis
dis.dis('( 7 * (119 - 23 ) ) / 12 + 16 / ( 3 + 1 )')
```

課題 5.15

解答省略。課題 5.14 のプログラムで後置記法の数式を入力して実行してください。

5.6 ソート（sort，整列）

課題 5.16

```
import string
import random
#random.seed(100)
def selection_sort(rlist):                                  #  選択ソートの定義
  for i in range(0, len(rlist) - 1):
    max = i                                                                  # 降順
    for j in range(i+1, len(rlist)):
      if rlist[max] < rlist[j]:
        max = j
    rlist[i], rlist[max] = rlist[max], rlist[i]
    print(i+1, 'after sorintg', rlist)
  return rlist
n = 12
rlist = [random.choice(string.ascii_letters) for i in range(n)]
print('before sort: random ', rlist)
print('after decending sort', selection_sort(rlist))               # 降順
print('use sorted function ', sorted(rlist, reverse = True))
```

課題 5.17*

解答省略（各自に任せる）

5.7 貪欲法と動的計画法

課題 5.18

```
def fibo(n):
  if n <= 1:
    return dp
  for i in range(2, n+1):            # dp table を用いて部分問題の解をメモ化する繰り返し
    dp[i] = dp[i-1] + dp[i-2]        # 再帰関係式で，再利用
  return dp                          # dp table を返す

n = int(input('Enter an integer n (>=0) for Fn: '))
dp = [0] * (n+1)                     # dp table の初期化
if n >= 1:
  dp[1] = 1
print(fibo(n))
```

i >= 2 の dp[i] は，メモ化している計算結果を再利用して求めることで，無駄な再計算を省いて高速化を図っています。再帰関数を用いてメモ化再帰する場合については，前著 p157 例題 6.14 を参照してください。

課題 5.19

```
dict_coins = {100: 0, 80: 0, 50: 0, 10: 0}
charge = int(input('Enter an amount_billed: '))       # 請求（支払い）金額
total_num_coins = 0                # the total number of used coins
total_paym = 0                      # total payments
for k in dict_coins.keys():
  q, r = divmod(charge, k)         # 商 q と余り r を返す組み込み関数
  dict_coins[k] = q                 # キーに対して値を入れる。coin： number
  total_num_coins += q
  total_paym += k * q
  charge = r
print('the total number of used coins by the greedy method:', total_num_coins)
print('dict_coins', dict_coins, '  total payments:', total_paym)
```

160 円の場合，最適解は得られません。490 円の場合には，最適解です。

課題 5.20

```
def min_coin_change(amount, dict_coins, list_dp):
##  請求額を支払うための最小のコイン枚数を求める。
  for koin in dict_coins.keys():              # 各コインごとに最小枚数を更新
    for paymt in range(koin, amount + 1):
      list_dp[paymt] = min(list_dp[paymt], list_dp[paymt - koin] + 1)
  min_num_coins = list_dp[amount]            # 請求額に対する最適解の最小枚数を代入。
##  各コインについてその使用枚数を求める
  paymt = amount                               # 請求額を計算額に代入して，初期化する。
  for koin, vnum in dict_coins.items():      # 各コインの使用枚数を求める繰り返し。
    while (paymt >= koin) and (list_dp[paymt] == list_dp[paymt-koin] + 1):
      vnum += 1                                # あるコイン koin の使用枚数をカウント +1
      paymt -= koin                            # paymt の更新。
    dict_coins[koin] = vnum                  # 辞書のキー koin に対して値 vnum を代入
  return min_num_coins, dict_coins

## メイン部
dict_coins = {1: 0, 5: 0, 10: 0, 50: 0, 100: 0, 500: 0}   # 辞書の初期化
print('initial dict_coins:', dict_coins)
amount = int(input('Enter an amount(yen): '))   # 請求額の入力
list_dp = [0] + [float('inf')] * amount           # list_dp の初期化：最小枚数の設定。
min_num_coins, dict_coins = min_coin_change(amount, dict_coins, list_dp)
print('the minimum number of used coins by the DP:', min_num_coins)
print('final dict_coins', dict_coins)                    # コインの辞書を出力
total = 0
for k, v in  dict_coins.items():
  total += k * v
print('the total payments(yen):', total)          # 全支払額の算出(請求額との一致)
```

実行結果

```
initial dict_coins {1: 0, 5: 0, 10: 0, 50: 0, 100: 0, 500: 0}
Enter an amount(yen): 5467
the minimum number of used coins: 19
final dict_coins {1: 2, 5: 1, 10: 1, 50: 1, 100: 4, 500: 10}
the total payments(yen): 5467
```

課題 5.21

課題 5.20 のプログラムで，硬貨の体系の初期化辞書と支払金額を設定して実行すればよい。

課題 5.22*

動的計画法：例題 5.15 のプログラムで，L24 と L28 のコードを変更して実行します。
貪欲法：例題 5.14 ののプログラムで，L24 と L25 のコードを変更して変更して実行します。

実行結果： 動的計画法

```
list_goods [(50, 5), (240, 20), (120, 20), (350, 50), (270, 30)] 5
the optimal solution (the max value): 630
selected dict_goods by the DP {5: (270, 30), 3: (120, 20), 2: (240, 20)} 3
value sum: 630    weight sum: 70    weight capacity: 70
```

実行結果：貪欲法

```
ndict_goods {2: (12.0, 240, 20), 1: (10.0, 50, 5), 5: (9.0, 270, 30), 4: (7.0, 350, 50),
3: (6.0, 120, 20)} 5
selected dict_goods by the greedy method {2: (12.0, 240, 20), 1: (10.0, 50, 5), 5: (9.0,
270, 30)} 3
value sum: 560    weight sum: 55    weight capacityt: 70
```

動的計画法のほうが，荷物を重量制限一杯に詰め込んでいて，価値も高く，優れているといえます。

5.8 解析結果のグラフ表現と線形化

課題 5.23

プログラム：グラフ 1

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(5, 8.0, 0.1)
y = np.array([632, 581, 469, 379, 306, 285, 217, 216, 160, 126, 109, 80,
 63, 48, 40, 34, 33, 23, 15, 14, 15, 12, 7, 2, 3, 4, 3, 3, 4, 2])
plt.scatter(x, y, s = 7)
plt.xlabel('magnitude')                          # マグニチュード
plt.ylabel('freqency')                           # 発生頻度
plt.show()
```

プログラム： グラフ 2

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(5, 8.0, 0.1)
y = np.array([632, 581, 469, 379, 306, 285, 217, 216, 160, 126, 109, 80,
 63, 48, 40, 34, 33, 23, 15, 14, 15, 12, 7, 2, 3, 4, 3, 3, 4, 2])
a, b = np.polyfit(x, np.log10(y), 1)
r = np.corrcoef(x, np.log10(y))[0, 1]
print('傾き a：', round(a, 3), '  y 切片 b：', round(b, 2),
```

```
      '  相関係数 r : ', round(r, 3), '  決定係数 r*r : ', round(r*r, 3))
y1 =10**(a*x + b)                          # log10 y1 = a * x + b
plt.plot(x, y1, 'red')
plt.scatter(x, y)
plt.yscale('log')
plt.xlabel('magnitude')
plt.ylabel('freqency')
plt.ylim(10**0, 10**3)
plt.show()
```

課題 5.24＊

プログラム： グラフ 1

```
import numpy as np
import matplotlib.pyplot as plt
x = []
for M in np.arange(5, 8.0, 0.1):
  x.append(int(10**(4.8 + 1.5 * M)))
y = np.array([632, 581, 469, 379, 306, 285, 217, 216, 160, 126, 109, 80,
 63, 48, 40, 34, 33, 23, 15, 14, 15, 12, 7, 2, 3, 4, 3, 3, 4, 2])
plt.scatter(x, y)
plt.xlabel('energy')                           # エネルギー
plt.ylabel('freqency')                         # 発生頻度
plt.show()
```

プログラム： グラフ 2

```
import numpy as np
import matplotlib.pyplot as plt
x = []
for M in np.arange(5, 8.0, 0.1):
  x.append(int(10**(4.8 + 1.5 * M)))
y = np.array([632, 581, 469, 379, 306, 285, 217, 216, 160, 126, 109, 80,
 63, 48, 40, 34, 33, 23, 15, 14, 15, 12, 7, 2, 3, 4, 3, 3, 4, 2])

A, B = np.polyfit(np.log10(x), np.log10(y), 1)
r = np.corrcoef(np.log10(x), np.log10(y))[0, 1]
print('傾き A:', round(A, 3), '  y 切片 B:', round(B, 2),
     '  相関係数 r:', round(r, 3), '  決定係数 r*r:', round(r*r, 3))
y1 = (10**B)*(x**A)                        # log10 y1 = A * log10 x + B

plt.plot(x, y1, 'red')
plt.scatter(x, y)
plt.xscale('log')
plt.yscale('log')
```

```
plt.xlabel('energy')
plt.ylabel('freqency')
plt.xlim(10**12, 10**17)
plt.ylim(10**0, 10**3)
plt.show()
```

課題 5.25

```
import numpy as np
import matplotlib.pyplot as plt
y = [485, 250, 195, 153, 146, 137, 110, 132, 125, 93, 103, 116,
     80, 67, 75,88, 83, 50, 58, 53, 53, 39, 60, 42, 44, 37, 47,
     73, 34, 38, 192, 188, 93, 83, 51, 38, 45, 51, 41, 30, 38,
     34, 27, 43, 26, 50, 29, 23, 24, 27, 32, 27,23, 26, 25, 31,
     11, 23, 29, 23, 32, 13, 27, 24, 26, 12, 16, 21, 15, 22, 19,
     15, 13, 21, 33, 24, 28, 21, 25, 22, 12]
y = np.array(y)
x = np.linspace(1, len(y), len(y))
a, b = np.polyfit(np.log10(x), np.log10(y), 1)
r = np.corrcoef(np.log10(x), np.log10(y))[0, 1]
print('傾き：', round(a, 3), '  y 切片：', round(b, 3),
      '  相関係数：', round(r, 3), '  決定係数：', round(r*r, 3))
plt.scatter(x, y, s = 15)
y2 = (x**a) * (10**b)
plt.plot(x, y2, 'red', lw = 2)
plt.xscale('log')                       # 両軸とも対数スケール
plt.yscale('log')                       # 大森公式のプロット
plt.show()
```

課題 5.26

プログラム： グラフ 1

```
import csv
import numpy as np
import matplotlib.pyplot as plt
with open('city_population.csv', 'r') as fobj:
  h = next(csv.reader(fobj))            # 最初の行のヘッダーを読む
  robj = csv.reader(fobj)
  list_data = [row for row in robj]
x, y = [], []
for i in range(len(list_data)):
  x.append(int(list_data[i][0]))
  y.append(int(list_data[i][1]))
x, y = np.array(x), np.array(y)
```

```
plt.scatter(x, y, marker = 'o', s = 7)
plt.xlim(0, 22)
plt.ylim(0, 5000)
listx = list(range(0, 22, 1))
plt.xticks(listx)
plt.xlabel('rank')
plt.ylabel('population')
plt.show()
```

プログラム： グラフ 2（政令指定都市の人口と順位）

```
import csv
import numpy as np
import matplotlib.pyplot as plt
with open('jpn_city_population.csv', 'r') as fobj:
  h = next(csv.reader(fobj))          # 最初の行のヘッダーを読む
  robj = csv.reader(fobj)
  list_data = [row for row in robj]
x, y = [], []
for i in range(len(list_data)):
  x.append(int(list_data[i][0]))
  y.append(int(list_data[i][1]))
x, y = np.array(x), np.array(y)
plt.scatter(x, y, marker = 'o', s = 6)
a, b = np.polyfit(np.log10(x), np.log10(y), 1)
r = np.corrcoef(np.log10(x), np.log10(y))[0, 1]
print('傾き a: ', round(a, 3), '  y 切片 b: ', round(b, 2),
      '  相関係数 r: ', round(r, 3), '  決定係数 r*r: ', round(r*r, 3))
y1 = (x**a) * (10**b)
plt.plot(x, y1, 'red')
plt.xscale('log')                # x 軸は log スケール
plt.yscale('log')                # y 軸は log スケール
plt.xlabel('rank')
plt.ylabel('population')
plt.show()
```

課題 5.27

```
import numpy as np
import matplotlib.pyplot as plt
a = [0.389, 0.724, 1.00, 1.524, 5.20, 9.510 ]                # saize 6
T = [0.24, 0.615, 1, 1.881, 11.862, 29.457]
x = np.array(a)
y = np.array(T)
A, B = np.polyfit(np.log10(x), np.log10(y), 1)
```

```
print('傾き A', round(A, 3), '\t', 'y 切片 B：', round(B, 3)))
y1 = (10**B)*(x**A)
plt.scatter(x, y)
plt.plot(x, y1, 'red')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('semi-major axis a')
plt.ylabel('period T')
plt.xlim(0.1, 100)
plt.ylim(0.1, 100)
plt.show()
```

8 つの惑星に関する解析は省略。

第 6 章　シミュレーションのプログラミング

6.1　放射性核種の半減期をシミュレーションにより求める

課題 6.1*

モジュールファイル　RI_decay.py

```
import random
def decay_data(seed_num, N0, trials, secs):  # データを生成する前処理の関数
  random.seed(seed_num)
  list_Nd = []
  for k in range(trials):
    N = N0
    Nd = []                                  # 崩壊後の核種の数を要素とするリスト
    Nd.append(N0)                            # 初期化
    for i in range(1, secs):                 # 繰り返しは，secs-1 秒後まで
      count1 = 0
      for j in range(N):                     # N 個のサイコロを振る繰り返し
        x = random.randint(1, 6)
        if x == 1:
          count1 += 1                        # 1 の目が出る個数のカウント
      N = N - count1                         # 崩壊した後の個数
      Nd.append(N)
    list_Nd.append(Nd)        # Nd の要素の数は secs。list_Nd の要素の数は trials
  data = []                    # trials で平均した核種の数を求め，それを要素とするリスト
  for i in range(secs):
    sumc = 0
    for j in range(trials):
      sumc += list_Nd[j][i]
```

```
    mean = round(sumc / trials, 1)       # 平均値を求める
    data.append(mean)
  return data                                  # 分析対象の崩壊データをリストとして返す


import numpy as np
import matplotlib.pyplot as plt
import RI_decay
seed_num = 100
N0 = 1000
trials = 10
secs = 16
data = RI_decay.decay_data(seed_num, N0, trials, secs)
print(len(data), data)
x = np.arange(0, secs)
y = np.array(data)
fig, ax = plt.subplots()
ax.set_xticks(np.arange(0, secs, 1))
ax.set_yticks(np.arange(0, 1100, 100))
ax.grid()
ax.scatter(x, y)
plt.show()
# 以下，y 軸を常用対数軸にしてグラフを描き，回帰係数と半減期を求める。
x = np.arange(0, secs)
y = np.array(data)
fig, ax = plt.subplots()                          # Matplotlib ax 系を用いる
ax.set_xticks(np.arange(0, secs, 1))
ax.grid()
a, b = np.polyfit(x, np.log10(y), 1)
r = np.corrcoef(x, np.log10(y))[0, 1]
print('a:', round(a, 4), '   b:', round(b, 2), '   r:', round(r, 3))
half_life = -np.log10(2) / a                      # 半減期
decay_const = np.log(2) / half_life   # decay_connst = (-a) * np.log(10) でも OK
print('Half_Life:', round(half_life, 2), '   decay_const:', round(decay_const, 3))
y1 = 10**(a*x + b)
ax.plot(x, y1, 'red')
ax.scatter(x, y)
plt.yscale('log')
plt.show()
```

実行結果

```
16 [1000.0, 832.3, 693.5, 587.1, 493.9, 410.8, 340.5, 283.9, 233.7, 195.8, 161.3, 136.8,
112.8, 95.3, 80.2, 66.7]
a: -0.0787    b: 3.0    r: -1.0
Half_Life: 3.83    decay_const: 0.181
```

プログラムでの常用対数 log10() と自然対数 log() の表記に注意。シミュレーションから得られた半減期の値が，計算値 $T = 3.80$ にかなり近い値であることが分かります。

グラフ化では軸の目盛りを細かく決めるために， ax.系の流儀を用いています。plt.系の流儀ではグラフの細かな調整ができないからです。

この放射性同位体の半減期は，放射性物質の人体や環境への影響を考えるときに欠かせない指標です。たとえば，セシウム 134 では半減期は 2.1 年ですが，ストロンチウム 90 では 28.7 年，プルトニウム 239 は 2.4 万年となります。

課題 6.2

```
import numpy as np
import matplotlib.pyplot as plt
T = 2.1
decay_const = np.log(2) / T                          # 崩壊定数λを求める。
print('崩壊定数 λ: ', round(decay_const, 3))     # 0.33
rdecay = []
for t in range(0, 11):
  activity = round(np.exp(-decay_const * t), 3)
  rdecay. append(activity)
print('減衰の割合のリスト: ', rdecay)
x = np.linspace(0, 10, num =11)
y = np.array(rdecay)
plt.plot(x, y , marker = 'o')
plt.xticks(np.arange(0, 11, 1))
plt.yticks(np.arange(0, 1.1, 0.1))
plt.xlabel('year')
plt.ylabel('radioactivity')
plt.show()
```

y 軸が log scale のグラフは，コード plt.yscale('log') を plt.show の前に挿入すればよい。

6.3 物体の放物運動をシミュレーションする

課題 6.3*

```
from math import *
import numpy as np
import matplotlib.pyplot as plt

def draw_trajectory(v0, theta, g):
  dt = 0.01
  time_flight = 2*v0*sin(theta)/g
```

```
  x = []
  y = []
  for t in np.arange(0, time_flight+dt, dt):  # 物体を投射してからの時間。時間の刻みは dt
    x.append(v0*cos(theta)*t)                 # 媒介変数 t
    y.append(v0*sin(theta)*t - 0.5*g*t*t)
  return x, y

g = 9.807
v0, angle1 = 30, 45
theta1 = radians(angle1)
x, y = draw_trajectory(v0, theta1, g)
plt.plot(x, y)
v0, angle2 = 30, 60
theta2 = radians(angle2)
x1, y1 = draw_trajectory(v0, theta2, g)
plt.plot(x1, y1)
plt.title('projectile motion')                  # グラフのタイトル
plt.xlabel('distance')                          # x 軸ラベル
plt.ylabel('height')                            # y 軸ラベル
plt.xlim(0, 100)
plt.ylim(0, 40)
plt.show()

print('Initial Velocity:', v0)
print('Project Angle:', angle1, '  max height:',
      round(0.5*v0*v0*sin(theta1)*sin(theta1)/g, 2),
      '  max distance:', round(v0*v0*sin(2*theta1)/g, 2))
print('Project Angle:', angle2, '  max height:',
      round(0.5*v0*v0*sin(theta2)*sin(theta2)/g, 2),
      '  max distance:', round(v0*v0*sin(2*theta2)/g, 2))
```

実行結果

```
Initial Velocity: 30
Project Angle: 45   max height: 22.94   max distance: 91.77
Project Angle: 60   max height: 34.41   max distance: 79.48
```

6.4 ロジスティック方程式のシミュレーション

課題 6.4*

```
import numpy as np
import matplotlib.pyplot as plt
r, K, N0 = 0.01, 1000, 10            # 内的自然増加率，環境収容数個体数の初期値
```

```
x = np.arange(0, 1001, 1)              # 離散的な間隔をとる時間
y = K * N0 * np.exp(r*x) / (N0 * np.exp(r*x) + K - N0)
plt.plot(x, y)
plt.title('increasing numbers of a living organism')
plt.xlabel('time')
plt.ylabel('numbers')
plt.grid()
plt.show()
```

$N(t)$ は，個体数が急速に増加した後，ある時点から増加が鈍化する S 字型の曲線を描き，最終的に環境収容力Kの値に収束します。ロジスティック曲線の傾きは，個体数の増加率 dN/dt を表します。傾きが変化する様子を図から読み取ってください。ロジスティック曲線は，家電製品の普及率などにもみられます。

課題 6.5

```
import numpy as np
import matplotlib.pyplot as plt
r, K , N = 0.01, 1000, [10]      # 内的自然増加率，環境収容数，個体数のリストの初期値
for i in range(0, 1000):
   N.append(N[i]+(r*N[i] - r*N[i]*N[i] / K))     # 離散時間での N の変化をリストで表す
x = np.arange(0, 1001, 1)
y = np.array(N)
plt.plot(x, y)
plt.title('increasing numbers of a living organism')
plt.xlabel('time')
plt.ylabel('numbers')
plt.grid()
plt.show()
```

6.5 ロジスティック写像のシミュレーション

課題 6.6*

```
import matplotlib.pyplot as plt
import numpy as np
for a in [0.5, 1.5, 2.5, 3.5, 4.0]:        # 5 個のリターンマップを描く
  x0 = 0.3                                 # 初期値。固定
  print('a:', a, '  x0:', x0)
  x = np. arange(0, 50.1, 0.1)
  y = x
  plt.plot(x, y, color = 'black')          # y = x の直線を描く
```

```
  x1 = np.arange(0, 100.01, 0.01)
  y1 = []
  for xi in x1:
    y1.append(a * xi * (1 - xi))
  y1 = np.array(y1)
  plt.plot(x1, y1, color = 'blue')          # y = ax(1-x)  放物線を描く

  x2= []
  x2.append(x0)                             # 初期値 x0
  for i in range(51):
    x2.append(a * x2[i] * (1 - x2[i]))

  x3 = []                                    # クモの巣図の座標の初期化
  x3.append(x2[0])
  x3.append(x2[0])
  y3 = []
  y3.append(0)
  for i in x2[1:]:
    x3.append(i)
    x3.append(i)
    y3.append(i)
    y3.append(i)
  x3.pop()                                   # 最後の要素は削除
  x3 = np.array(x3)
  y3 = np. array(y3)
  plt.plot(x3, y3, color = 'red')            # クモの巣図を描く

  plt.xlim(0,1)
  plt.ylim(0, 1)
  plt.xlabel('Xn')
  plt.ylabel('Xn+1')
  plt.show()
```

課題 6.7＊

```
import matplotlib.pyplot as plt
import numpy as np
def x(n):
  global a
  if n == 0:
    return 0.3                              # x0 = 0.3
  else:
    if n not in memo.keys():
```

```
      memo[n] = a*x(n-1)*(1 - x(n-1))    # メモ化再帰
    return memo[n]

m = 51
for a in [0.5, 1.5, 2.5, 3.5, 4.0]:       # 5 個の n_xn の図を描く
  list_x = []
  memo = {}
  for i in range(0, m):
    list_x.append(x(i))

  n = np.arange(0, 51)
  X = np.array(list_x)
  print('a:', a, '   X[0]:', X[0])
  plt.plot(n, X, marker ='o', markersize = 5)
  plt.xlim(0, 51)
  plt.ylim(0, 1)
  plt.xlabel('n')
  plt.ylabel('Xn')
  plt.show()
  print()
```

第 7 章　オブジェクト指向プログラミング（OOP）

7.1　クラスを自作し，モジュール化する

課題 7.1

モジュールファイル　body_indices.py

```
class MyBody:                                    # クラスの定義
  def __init__(self, height, weight):
    self.height = height
    self.weight = weight
  def bmi_sw_dod(self):
    h2 = self.height*self.height
    bmi = self.weight / h2
    sw = h2*22
    dod = (self.weight-sw)/sw *100
    return bmi, sw, dod


import body_indices                                  # モジュールのインポート
ht = float(input('身長(cm)を入力してください：')) / 100
wt = float(input('体重(kg)を入力してください：'))
```

```
mybody = body_indices.MyBody(ht, wt)              # モジュールのクラスの呼び出し
bmi, sw, dod = mybody.bmi_sw_dod()                # モジュールのメソッドの呼び出し
print('BMI:', round(bmi, 1))
print('標準体重 Standard Weight:', round(sw, 1))
print('肥満度 Degree of Obesity:', round(dod, 1))
```

課題 7.2

```
class GcdEuclid:                # クラスの定義
  def __init__(self, x, y):    # 初期化メソッドの定義
    self.x = x
    self.y = y
  def gcd(self):                # メソッドの定義
    while self.y != 0:
      self.x, self.y = self.y, self.x % self.y
    return self.x

x, y =  map(int, input('Enter two integers: ').split())
gcdob = GcdEuclid(x, y)         # オブジェクトの生成
print('GCD:', gcdob.gcd())      # メソッドを呼び出す
```

課題 7.3

モジュールフィアル　GcdEuclid.py

```
class GcdEuclid:
  def __init__(self, x, y):
    self.x = x
    self.y = y
  def gcd(self):
    while self.y != 0:
      self.x, self.y = self.y, self.x % self.y
    return self.x
```

```
import GcdEuclid                    # モジュールファイルのインポート
x, y =  map(int, input('Enter two integers:  ').split())
gcdob = GcdEuclid.GcdEuclid(x, y)     # モジュールファイルに収められたクラスを呼び出す
print('GCD:', gcdob.gcd())            # オブジェクトがメソッドを呼び出す
```

課題 7.4

モジュールファイル　QuadraEq.py

```
class QuadraEq:                                  # クラスの定義
  def __init__(self, a, b, c):
    self.a, self.b, self.c = a, b, c
  def solutions(self):
    D = self.b**2 - 4*self.a*self.c
    sqrd = D**0.5
    x1 = (-self.b + sqrd) / (2*self.a)
    x2 = (-self.b - sqrd) / (2*self.a)
    return x1, x2


import QuadraEq                  # モジュールファイルのインポート
a, b, c = 1, -2, -3
obj1 = QuadraEq.QuadraEq(a, b, c)              # obj1 の生成
print('2 次方程式の解 ', obj1.solutions())
print(type(obj1))                              # <class 'QuadraEq.QuadraEq'>

a, b, c = 2, -5, 4
obj2 = QuadraEq.QuadraEq(a, b, c)               # obj2 の生成
print('2 次方程式の解 ', obj2.solutions())
print(isinstance(obj2, QuadraEq.QuadraEq))      # True
```

組み込み関数 isinstance()は，第 1 引数のオブジェクトが，第 2 引数の型（クラス）のインスタンスであれば True を返します。

課題 7.5

```
class Fibonacci:                               # クラスの定義
  def __init__(self, x = 0, y = 1):       #  初期化メソッド。初期値あり。
    self.x = x                                  # インスタンス変数の初期化
    self.y = y
  def fibo(self, n):
    self.x, self.y = 0, 1
    for i in range(n):
      self.x, self.y = self.y, self.x + self.y
    return self.y

fib_obj = Fibonacci()                                  # オブジェクトの生成。引き数なし
print([fib_obj.fibo(i) for i in range(20)])
```

課題 7.6

```
class Stack:                             # クラスの定義
  def __init__(self):
```

```
    self.stack = []
  def push(self, item):
    self.stack.append(item)
  def pop(self):
    return self.stack.pop()
  def get_list_rpn(self, rpn):
    list_str = rpn.split()
    rpn_queue = []
    for s in list_str:
      if s in '+-*/':
        rpn_queue.append(s)
      else:
        rpn_queue.append(eval(s))
    return rpn_queue
  def shisoku(self, i):
    print(i, '演算子が取り出されたので，二項演算を実行します')
    a = self.pop()
    b = self.pop()
    print('数値スタックでの数値の並び b a \n', b, a)
    print(' ', i)
    if i == '+':
      val= b + a
    elif i == '-':
      val = b - a
    elif i == '*':
      val = b * a
    elif i == '/':
      val = b / a
    return self.push(val)
  def stack_computing(self, rpn_queue):
    for i in rpn_queue:
      if type(i) is float or type(i) is int:
        self.push(i)
      else:
        self.shisoku(i)
      print('演算後のスタック', self.stack)
    return self.pop()

#  メイン部
rpn = input('後置記法の数式を入力してください： ')
stack_compute = Stack()                          # クラスを呼び出してオブジェクトの生成
rpn_queue = stack_compute.get_list_rpn(rpn)
print('rpn_queue', rpn_queue)
print('計算結果 ', stack_compute.stack_computing(rpn_queue))
```

課題 7.7＊

```
import convert
import stack_computer
infix = input('Enter an infix exp: ')
list_infix = infix.split()
cnvobj = convert2.Convert(list_infix)
rpn_queue = cnvobj.infix_rpn()
rpn = ' '.join(rpn_queue)
print('rpn exp ', rpn)
stack = []
stack_comp = stack_computer2.Stack(rpn, stack)
print('result =', stack_comp.stack_computing())
```

モジュールファイル　convert.py

```
def comp(self, b, a, op_stack, rpn_queue):
    if b + a in ['+*', '+/', '-*', '-/']:
      op_stack.append(b)
      op_stack.append(a)
    else:
      rpn_queue.append(b)
      op_stack.append(a)
    return op_stack, rpn_queue

class Convert:
  def __init__(self, list_infix):
    self.list_infix = list_infix
  def infix_rpn(self):
    rpn_queue = []
    op_stack = []
    for c in self.list_infix:
      if c in '()+-*/':
        if c == '(':
          op_stack.append(c)
          continue
        if c == ')':
          op_stack.append(c)
          op_stack.pop()
          rpn_queue.append(op_stack.pop())
          x = op_stack.pop()
          if x == '(':
            continue
          else:
            rpn_queue.append(x)
            if op_stack.pop() == '(':
              continue
```

```
        op_stack.append(c)
        if len(op_stack) >= 2:
          a = op_stack.pop()
          b = op_stack.pop()
          if b == '(':
            op_stack.append(b)
            op_stack.append(a)
            continue
          if b != '(':
            self.comp(b, a, op_stack, rpn_queue)
            if len(op_stack) == 2:
              a = op_stack.pop()
              b = op_stack.pop()
              if b == '(':
                op_stack.append(b)
                op_stack.append(a)
                continue
              self.comp(b, a, op_stack, rpn_queue)
      else:
        rpn_queue.append(c)
    if len(op_stack) == 2:
      rpn_queue.append(op_stack.pop())
      rpn_queue.append(op_stack.pop())
    elif len(op_stack) == 1:
      rpn_queue.append(op_stack.pop())
    return rpn_queue
```

モジュールファイル　stack_computer.py

```
def shisoku(self, i):
  a = self.pop()
  b = self.pop()
  if i == '+':
    val= b + a
  elif i == '-':
    val = b - a
  elif i == '*':
    val = b * a
  elif i == '/':
    val = b / a
  return self.push(val)

class Stack:
  def __init__(self, rpn, stack):
    self.rpn = rpn
    self.stack = stack
  def push(self, elm):
```

```
    return self.stack.append(elm)
  def pop(self):
    return self.stack.pop()

  def get_list_rpn(self):
    list_str = self.rpn.split()
    list_rpn = []
    for c in list_str:
      if c in '+-*/':
        list_rpn.append(c)
      else:
        list_rpn.append(eval(c))
    return list_rpn
  def stack_computing(self):
    list_rpn = self.get_list_rpn()
    for i in list_rpn:
      if type(i) is float or type(i) is int:
        self.push(i)
      else:
        self.shisoku(i)
    return self.pop()
```

課題 7.8＊

```
import get_emjdata5
import TestScore_Pass
e, m, j = get_emjdata5.emjdata5()                    #  科目別データ  e, m, j
tsobj = TestScore_Pass.TestScore_Pass(e, m, j)       #オブジェクトの生成
tsobj.class_const()                                  # __PASS_SCORE  60
print(tsobj.count_pass())                            # [8, 7, 10]
```

モジュールファイル　get_emjdata5.py

```
def emjdata5():
  import csv
  with open('test_score5.csv', 'r') as fobj:
    h = next(csv.reader(fobj))
    rdobj = csv.reader(fobj)
    data = []
    for row in rdobj:
      dat = []
      for i in row:
        if i.isdigit():
          dat.append(float(i))    # 数字なら float 型に変換
        else:
          dat.append(i)
```

```
      data.append(tuple(dat))
# 科目別データに分ける
  e, m, j = [], [], []
  for i in range(len(data)):
    e.append(data[i][2])
    m.append(data[i][3])
    j.append(data[i][4])
  return e, m, j
```

モジュールファイル　TestScore_Pass.py

```
class TestScore_Pass:                          # クラスの定義
  __PASS_SCORE = 60        # クラス定数
  def __init__(self, eng, math, jpn):
    self.eng = eng
    self.math = math
    self.jpn = jpn
  def count_pass(self):
    length = len(self.eng)
    count = []
    for subj in [self.eng, self.math, self.jpn]:
      pcount = 0
      for j in range(length):
       if subj[j] >= TestScore_Pass.__PASS_SCORE:  # クラス変数の使用
         pcount += 1
      count.append(pcount)   # 科目ごとの合格者数がリスト count の要素
    return count             # リストで返す
  @classmethod
  def class_const(cls):
    return print('__PASS_SCORE ', cls.__PASS_SCORE)
```

課題 7.9

```
class BankAccount:
    def __init__(self, money):
        self.money = money                # money 属性
    def deposit(self, dp_money):
        self.money += dp_money
    def withdraw(self, wd_money):
        if wd_money <= self.money:
            self.money -= wd_money
            flag = True
        else:
            flag = False
        return flag
```

```
# メインのプログラム
it_money = int(input('Enter initial money: '))
account = BankAccount(it_money)            # BnkAccount クラスのオブジェクト account の生成
print(' Initial money:', account.money)    # オブジェクト account がデータ属性 money を参照
while True:
  wd_money = int(input('Enter withdrawal_money: '))
  if wd_money == 0:
    print(' End of Processing')
    break
  flag = account.withdraw(wd_money)         # account がメソッド withdraw にアクセス
  if flag == True:
    print(' Money after withdrawal:', account.money)
  else:
    print(' Warning: Money is insuffcient!')
  dp_money = int(input('Enter deposit_money: '))
  if dp_money == 0:
    print(' End of Processing')
    break
  account.deposit(dp_money)                 # account がメソッド deposit にアクセス
  print(' Money after deposit:', account.money)
```

7.2 継承（inheritance）と多態性（polymorphism）

課題 7.10

```
import math
class Circle:                           # 基底クラス
  def __init__(self, r):
    self.r = r
  def area(self):
    s = math.pi * self.r **2
    return s
class Cylinder(Circle):                 #  派生クラス。Circle クラスの継承
  def __init__(self, r, h):
    super().__init__(r)
    self.h = h
  def vol_cyl(self):
    V = self.area() * self.h
    return V
class Cone(Cylinder):                   #  派生クラス。Cylinder クラスの継承
  def vol_cone(self):
    V = self.vol_cyl() / 3
    return V
```

```
#  メイン部（クラス外）
crobj = Circle(10)                          # クラスからオブジェクトの生成
print('半径：', crobj.r)
ar = crobj.area()                           # オブジェクトがメソッドにアクセスして
print('円の面積：', round(ar, 2))
cylobj = Cylinder(10, 20)
print('高さ：', cylobj.h)
print('円柱の体積：', round(cylobj.vol_cyl(), 2))
cnobj = Cone(10, 20)
print('円錐の体積：', round(cnobj.vol_cone(), 2))
```

課題 7.11

```
import math
class Sides:                                        # 基底クラス
  COEFF = 25.4
  def __init__(self, diagonal, aspect_ratio):
    self.c = diagonal * Sides.COEFF
    self.ratio = aspect_ratio
  def width_height(self):
    h = self.c / math.sqrt(1 + self.ratio * self.ratio)
    w = self.ratio * h
    return w, h
class Area(Sides):                                  # 派生クラス。Sides クラスの継承
  def area(self):
    w, h = self.width_height()
    s = h/10 * w/10
    return s

# プログラム本体（クラスの外）
diagonal = 12.9
print('diagonal:', diagonal, 'inch')
aspect_ratio = 4/3
print('aspect_ratio 4/3:', round(aspect_ratio, 2))
side = Sides(diagonal, aspect_ratio)                #  クラスからオブジェクトの生成
w, h = side.width_height()
print('width:', round(w, 1), 'mm')
print('height:', round(h, 1), 'mm')
s = Area(diagonal, aspect_ratio)                    #  クラスからオブジェクトの生成
print('area:', round(s.area(), 1), 'square cm')
```

課題 7.12*

```
class Pangram(str):                    # str クラスを継承する派生クラスの定義
  def count_letters(self):             # メソッドの定義
    stringx = self.replace(' ', '').replace('.', '').lower()
    dict_count = {}                    # アルファベットの辞書
    for c in stringx:                  # c は辞書のキー
      if c in dict_count:
        dict_count[c] += 1
      else:
        dict_count[c] = 1
    if len(dict_count) == 26:          # 辞書のサイズが 26 かどうか
      flag = True                       # すべての文字が使われている。パングラム
    else:
      flag = False                      # 使われていない文字がある
    return stringx, len(stringx), len(dict_count), flag, dict_count

#  クラスの外
sentence = Pangram('The quick brown fox jumps over a lazy dog.')
print(sentence)
stringx, total_letters, diff_letters, flag, dict_count\
 = sentence.count_letters()                  # 派生クラスのメソッドを呼び出す
print(''.join(sorted(stringx)) )
print('total_letters:', total_letters)      # 使われた全文字数
print('diff_letters:', diff_letters)        # 異なった文字数
print('flag:', flag)
import pprint
pprint.pprint(dict_count)
print(sentence.upper())       # str クラスのメソッドの使用
print(sentence.split())       # str クラスのメソッドの使用
```

課題 7.13

```
import abc
class Animal(metaclass = abc.ABCMeta):     # 抽象クラスの定義
  @abc.abstractmethod
  def make_cry(self):
    pass
  def behavior(self):
    pass
class Dog(Animal):                          # 継承。具象クラスの定義
  def make_cry(self):
    return 'bowwow'
  def behavior(self):
    return 'depemdent'
class Cat(Animal):                          # 継承。具象クラスの定義
  def make_cry(self):
```

```
    return 'meow'
  def behavior(self):
    return 'indepemdent'

dog, cat = Dog(), Cat()                          # オブジェクトの生成
for animal in [dog, cat]:
  print(type(animal), '  Cry:', animal.make_cry())
  print(type(animal), '  Behavior:', animal.behavior())
  print()
```

7.3 クラスのオブジェクト同士の相互作用

課題 7.14*

```
class PetOwner:
  def __init__(self, name):
    self.name = name
  def message(self, cat):                  # cat 仮引数。仮オブジェクト
    print(self.name + ' feeds his cat ' + cat.name + ', ' \
    + cat.breed +'. ' + cat.meow())
class Cat:
  def __init__(self, name, breed):         # データ属性
    self.name = name
    self.breed = breed
  def meow(self):                          # メソッド属性
    return 'Meow!'

# メインのプログラム
cat = Cat('Luna', 'American Shorthair')    # Cat オブジェクト cat の生成
petowner = PetOwner('John')                # PetOwer オブジェクト petowner の生成
petowner.message(cat)                      # petowner が message メソッドを呼び出す
```

オブジェクト petowner がメソッド message を呼び出すとき，オブジェクト cat を引数として受け取っています。オブジェクト cat は cat.name，cat.breed によりデータ属性に，cat.meow によりメソッド属性にアクセスできますので，メソッド message の print 出力によりメッセージ

```
John feeds his cat Luna, American Shorthair. Meow!
```

が発せられることになります。このように，メッセージの発信は，2 つの異なるオブジェクトの相互作用（協力）によりなされていることが分かります。

第 8 章　タートルグラフィックス（TG）の OOP

8.2　TG_POP と TG_OOP

課題 8.1

```
import turtle
size = 150
n = 5
angle = 180 - 180 / n                # 角度の単位は度。外角 angle = 144
turtle.pencolor('red')               # 軌跡に色(赤)をつける
turtle.pensize(2)
for i in range(n):
    turtle.forward(size)
    turtle.right(angle)              # 外角の大きさで，右に向く
turtle.done()
```

課題 8.2

```
import turtle
def draw_squares(angle, length, add, m):
    if m == 10:
        return
    for i in range(4):
        kameo.forward(length)
        kameo.left(angle)
    return draw_squares(angle, length+add, add, m+1)

screen = turtle.Screen()                # screen オブジェクトの作成
screen.setup(600, 600)
screen.title('My Turtle Graphics')
kameo = turtle.Turtle()                 # kameo オブジェクトの作成
kameo.pencolor('red')
kameo.pensize(2)
kameo.speed(2)
angle, length, add, m = 90, 10, 10, 1
draw_squares(angle, length, add, m)
screen.mainloop()
```

課題 8.3

```
import turtle
def draw_spiral(size):
```

```
    if size > 30:
        return
    kameo.forward(size)
    kameo.right(15)
    return draw_spiral(size*1.02)    # 再帰呼び出し

screen = turtle.Screen()             # screen オブジェクトの作成
screen.setup(600, 600)
screen.title('My Turtle Graphics')
kameo = turtle.Turtle()              # kameo オブジェクトの作成
draw_spiral(5)                       # 関数呼び出し
screen.mainloop()
```

課題 8.4

```
import turtle
def draw_circles():                  # 円を描く関数の定義
    seven_colors = ['red', 'orange', 'yellow', 'green', 'blue',
                'indigo', 'violet']
    r = 30
    for i in seven_colors:
        kameo.pencolor(i)
        kameo.circle(r)
        r += 10

screen = turtle.Screen()             # screen オブジェクトの生成
screen.setup(600, 600)
screen.title('My Turtle Graphics')
kameo = turtle.Turtle('turtle')      # kameo オブジェクトの生成
kameo.pensize(3)
draw_circles()
screen.mainloop()
```

課題 8.5

```
import turtle
class Rainbow:                                  #  Rainbow クラスの定義
    def __init__(self, radius, decrease):
        self.radius = radius
        self.decrease = decrease
        self.y = 0
    def draw_semicircle(self, color):             # 半円を描くメソッド
        kameo.penup()
        kameo.setposition(self.radius, self.y)
```

```
        kameo.setheading(90)                    # 90 度の向き（北）
        kameo.pendown()
        kameo.pencolor(color)
        kameo.circle(self.radius, 180)          # 半円状の帯を描く
        self.radius -= self.decrease        # 半径を描いた色の分だけ減少させる

# クラスの外
screen = turtle.Screen()                # screen オブジェクトの生成
screen.setup(600, 600)
screen.title('My Turtle Graphics')
kameo = turtle.Turtle()                 #  kameo オブジェクトの生成
radius, decrease = 100, 10              #  半径と decrease（半径の減少分）
bow = Rainbow(radius, decrease)         # インスタンス化，オブジェクトを生成
kameo.pensize(decrease + 2)             # ペンの太さ。+2 は微調整，色の帯とする
seven_colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
for i in range(7):
    bow.draw_semicircle(seven_colors[i])   # メソッドを呼び出す
kameo.hideturtle()       # カメを消す
screen.mainloop()
```

課題 8.6

```
import turtle
wn = turtle.Screen()                    #  wn オブジェクト（スクリーン）の生成
wn.setup(800, 800)
wn.title('My Turtle Graphics')
kamesan = turtle.Turtle()               # kamesan オブジェクトの生成
kamesan.hideturtle()
kamesan.speed(0)

w = int(400)
h = int(w * 2/3)
r = int((h * 3/5) * 1/2)
kamesan.penup()
kamesan.goto(0, -r)
kamesan.pensize(2)
kamesan.pendown()
kamesan.color('red')
kamesan.begin_fill()
kamesan.circle(r, steps = 600)
kamesan.end_fill()
kamesan.penup()
kamesan.goto(-0.5*w, -0.5*h)
kamesan.setheading(0)
kamesan.color('black')
```

```
kamesan.pensize(1)
kamesan.pendown()
kamesan.forward(w)
kamesan.left(90)
kamesan.forward(h)
kamesan.left(90)
kamesan.forward(w)
kamesan.left(90)
kamesan.forward(h)

kamesan.penup()
kamesan.goto(0, 0.5*h+30)
kamesan.write('日章旗', align = 'center',
              font = ('MS Gothic', 20, 'bold'))
wn.mainloop()
```

課題 8.7

```
import turtle
def draw_rectangle(step1, step2, angle, color):    # 長方形の描画
    kamesan.color(color)
    kamesan.begin_fill()
    for count in range(2):
        kamesan.forward(step1)
        kamesan.right(angle)
        kamesan.forward(step2)
        kamesan.right(angle)

wn = turtle.Screen()                        # wn は creen でもよい
wn.setup(800, 800)
wn.title('My Turtle Graphics')
kamesan = turtle.Turtle()                   # kamesan オブジェクトの生成
kamesan.hideturtle()
kamesan.speed(1)
draw_rectangle(50, 100, 90, 'blue')
kamesan.end_fill()
kamesan.penup()
kamesan.goto(50, 0)
kamesan.pendown()
kamesan.color('black')
kamesan.forward(50)
draw_rectangle(50, 100, 90, 'red')
kamesan.end_fill()
kamesan.penup()
kamesan.goto(50, -100)
```

```
kamesan.down()
kamesan.color('black')
kamesan.forward(50)

kamesan.penup()
kamesan.goto(70, 30)
kamesan.color('black')
kamesan.write('The French Flag', align = 'center',
          font = ('Arial', 15, 'normal'))
wn.mainloop()
```

8.3 TG_OOP におけるイベント処理

課題 8.8

```
import turtle
def draw_square(x, y):
    for i in range(4):
        kameo.forward(200)
        kameo.left(90)
        kamea.right(90)
        kamea.forward(195)

screen = turtle.Screen()
screen.setup(700, 700)
screen.title('My Turtle On-Screen_Click Graphics')
kameo = turtle.Turtle()
kameo.shape('turtle')
kameo.pensize(3)
kameo.pencolor('blue')
kamea = turtle.Turtle()
kamea.shape('turtle')
kamea.pensize(3)
kamea.pencolor('red')
#kameo.home()                              # なくてもよい
#kamea.home()                              # なくてもよい
screen.onscreenclick(draw_square)         # スクリーンをマウスでクリックすれば作図を開始
screen.mainloop()
```

課題 8.9

```
import turtle
def motion(i):
```

```
    kameo.forward(1.745)
    kameo.left(1)
    kamea.forward(1.745)
    kamea.right(1)
    if i == 180:
        return
    return motion(i+1)
def encounter(x, y):
    kameo.pendown()
    kamea.pendown()
    motion(0)
def kameo_attribute():
    kameo.shape('turtle')
    kameo.color('blue')
    kameo.pensize(2)
def kamea_attribute():
    kamea.shape('turtle')
    kamea.color('red')
    kamea.pensize(2)

screen = turtle.Screen()
screen.setup(600, 700)
screen.title('An Enkounter of Kameo and Kamea')
kameo = turtle.Turtle()
kameo_attribute()
kameo.goto(16, 0)
kamea = turtle.Turtle()
kamea_attribute()
kamea.goto(-16, 0)
kamea.setheading(180)
screen.onscreenclick(encounter)    # スクリーンをマウスでクリックすれば作図を開始
screen.mainloop()
```

8.4 TG_OOP でフラクタル図形を描く

課題 8.10＊

```
def draw_tree(length, rate, angle, order):
    if order >= 0:
        kameo.forward(length)
        kameo.right(angle)
        draw_tree(length*rate, rate, angle, order-1)
        kameo.left(angle*2)
        draw_tree(length*rate, rate, angle, order-1)
```

```
        kameo.right(angle)
        kameo.backward(length)
    return

import turtle
screen = turtle.Screen()                    # screen オブジェクトの作製
screen.setup(1200, 900)
screen.title('My Turtle Graphics: A Fractal Tree')
kameo = turtle.Turtle()                     # kameo オブジェクトの作成
kameo.penup()
kameo.goto(0, -400)
kameo.pendown()
kameo.pencolor('green')                                 # 木を緑にする
kameo.pensize(3)
#kameo.speed(0)
screen.tracer(100)
kameo.left(90)                                         # 開始点で垂直上方に向きをとる
length, rate, angle, order = 150, 0.82, 30, 12         # 描画パラメータ
print('枝の長さの最小値： ', int(length*rate**order))    # 13
draw_tree(length, rate, angle, order)                 # 作図関数の呼び出し
screen.mainloop()
```

課題 8.11＊

```
def Koch_curve(length, order):                    # Koch 曲線
    if order == 0:
        kameo.forward(length)
    elif order >= 1:
        for angle in [0, 60, -120, 60]:           # angle 描画パラメータ
            kameo.left(angle)
            Koch_curve(length/3, order - 1)

import turtle
screen = turtle.Screen()                    # オブジェクト screen の生成
screen.setup(600, 600)
screen.title('My Turtle Graphics: a Koch Snawflake Figure')
kameo = turtle.Turtle()                     # オブジェクト kameo の生成
kameo.hideturtle()
kameo.penup()
kameo.goto(-150, 100)
kameo.speed(0)
kameo.pendown()
kameo.pensize(2)
length, order = 300, 4                      # 描画パラメータ
```

```
for i in range(3):                          # 雪片を描く
    Koch_curve(length, order)
    kameo.right(120)                        # Koch curve を繋ぎ合わせる
screen.mainloop()
```

課題 8.12＊

```
def Sierpinski_triangle(length, angle, order):
    if order >= 0:
        for i in range(3):                      # 三角形を描く
            kameo.forward(length)
            kameo.left(angle)
            Sierpinski_triangle(length/2, angle, order - 1)

import turtle
screen = turtle.Screen()
screen.setup(600, 600)
screen.title('My Turtle Graphics: a Sierpinski_triangle Figure')
kameo = turtle.Turtle()                         # kameo オブジェクトを生成
kameo.hideturtle()
kameo.penup()
kameo.goto(-200, -100)
kameo.pendown()
kameo.pensize(2)
#kameo.speed(0)
screen.tracer(100)
length, angle, order = 300, 120, 5              ## 描画パラメータ
Sierpinski_triangle(length, angle, order)
screen.exitonclick()                # マウスをスクリーン上で左クリックしてプログラム終了
```

**課題 8.13*

```
from ColabTurtlePlus.Turtle import *
clearscreen()
def draw_tree(length, rate, limit, angle):
    if length >= limit:
        kameo.forward(length)
        kameo.right(angle)
        draw_tree(length*rate, rate, limit, angle)
        kameo.left(angle*2)
        draw_tree(length*rate, rate, limit, angle)
        kameo.right(angle)
        kameo.backward(length)
```

```
screen = Screen()
screen.setup(1000, 1000)
kameo = Turtle()                                    # kameo オブジェクトの作成
kameo.penup()
kameo.goto(0, -400)
kameo.pendown()
kameo.pensize(2)
kameo.pencolor('green')
kameo.speed(0)                                      #  最高速描画
kameo.left(90)
length, rate, limit, angle = 150, 0.82, 14, 30     # rate，limit を変えて図形の変化をみる
draw_tree(length, rate, limit, angle)
kameo.done()                                        #  speed(0) のときに必須
```

第 9 章　GUI アプリ作成の OOP

9.1　GUI アプリの作成に Tkinter を使う

課題 9.1

```
import tkinter as tk
##  Model（モデル部）: 算術式 を計算して値を出力する
# イベントハンドラー basic_calc(event) 関数を定義する
def basic_calc(values):
    x = float(x_box.get())
    y = float(y_box.get())
    op = op_box.get()
    if op == '+':
        z = x + y
    elif op == '-':
        z = x - y
    elif op == '*':
        z = x * y
    elif op == '/':
        z = x / y
    elif op == '//':
        z = x // y
    elif op == '%':
        z = x % y
    elif op == '**':
        z = x ** y
    else:
        z = 'invalid'
    result_box.delete(0, tk.END)
```

```
    result_box.insert(0, z)

def finish(event):
    root.destroy()

## View （外観部）
# ウインドウを作り，ラベル，ボックスを作り，これらを配置する
root= tk.Tk()
root.geometry('400x300')
root.title('Basic Calculator')
input_label = tk.Label(text = '二つの数を入力してください')
input_label.pack()
x_label= tk.Label(text = 'x')
x_label.pack()
x_box = tk.Entry()
x_box.pack()
y_label=tk.Label(text = 'y')
y_label.pack()
y_box = tk.Entry()
y_box.pack()
input_label = tk.Label(text = '演算子を +,  -,  *,  /,  //,  %,  **  \
の中から一つを選んでください')
input_label.pack()
op_label= tk.Label(text = '演算子')
op_label.pack()
op_box = tk.Entry()
op_box.pack()
calc_button = tk.Button()
calc_button['text'] = '計算ボタン'
calc_button.pack()
result_label= tk.Label(text = '結果')
result_label.pack()
result_box = tk.Entry()
result_box.pack()
fin_button = tk.Button()
fin_button['text'] = '終了ボタン'
fin_button.pack()

##  Controller（制御部）
# イベントとイベントハンドラーを結合させる
calc_button.bind('<Button-1>', basic_calc)        # マウスのクリック
fin_button.bind('<Button-1>', finish)
root.mainloop()
```

課題 9.2

```
import tkinter as tk
#  Model（モデル部）
def check_digit(event):
    ccn = x_box.get()
    ccn = ccn.replace(' ', '')
    figs = ccn[::-1]
  sum_odd, sum_even = 0, 0
  for i, j in enumerate(figs, 1):        # i は 1 から始まる。j はカード番号の数字
      if i % 2 != 0:                     # インデックスが奇数
          sum_odd += int(j)              # 数字を数に変換
      else:                              # インデックスが偶数
          n = int(j) * 2
          if n >= 10:
              sum_even += n - 9
          else:
              sum_even += n
  sum_ccn = sum_odd + sum_even
  if sum_ccn % 10 == 0:
      ccn_box.delete(0, tk.END)
      ccn_box.insert(0, 'True')
  else:
      ccn_box.delete(0, tk.END)
      ccn_box.insert(0, 'False')

def finish(event):
    root.destroy()

## View（外観部）
root= tk.Tk()
root.geometry('400x220')
root.title('Credit Card Number Checker')
input_label = tk.Label(text = 'クレジット番号を左端から 4 桁ごとに\
空白を 1 つ入れて入力してください')
input_label.pack()
x_label= tk.Label(text = 'Credit Card Number')
x_label.pack()
x_box = tk.Entry()
x_box.pack()
calc_button = tk.Button()
calc_button['text'] = '計算ボタン'
calc_button.pack()
ccn_label= tk.Label(text = 'Check Digit')
ccn_label.pack()
ccn_box = tk.Entry()
```

```
ccn_box.pack()
fin_button = tk.Button()
fin_button['text'] = '終了ボタン'
fin_button.pack()

##  Controller（制御部）
calc_button.bind('<Button-1>', check_digit)    # マウスの左クリック
fin_button.bind('<Button-1>', finish)
root.mainloop()
```

課題 9.3＊

```
import tkinter as tk
import tkinter.ttk as ttk              # このモジュールを必要とする

##  Model: GPAcalculation
gp_credit = 0
credits = 0
def GPA_handler(event):                    # 引数 event はないとエラーが起きる
    global gp_credit
    global credits
    subject = subject_box.get()              #  科目名
    credit = int(credit_box.get())           # 単位数
    RS = float(score_box.get())          # Raw Score（RS,素点）
    if RS > 100:
        print('入力エラー，点数を入れなおしてください！')
    LG, GP = LGGP_calc(RS)
    LG_box.delete(0, tk.END)
    LG_box.insert(0, LG)
    GP_box.delete(0, tk.END)
    GP_box.insert(0, GP)
    if RS <= 100.0:
        gp_credit += GP*credit                 # GPA の計算式の分子
        credits += credit                      # GPA の計算式の分母
        GPA = gp_credit / credits
        GPA = round(GPA, 2)
        GPA_box.delete(0, tk.END)
        GPA_box.insert(0, GPA)
        tree.insert('', 'end', values = (subject, credit, RS, LG, GP, GPA))
#    履修表に 1 行の成績レコードを書き込む

def fin_handler(event):                # プログラムを終了させるイベントハンドラーの定義
    root.destroy()

def LGGP_calc(RS):
```

```
    if 90 <= RS <=100:
        LG = 'S'
        GP = 4.0
    elif 80 <= RS < 90:
        LG = 'A'
        GP = 3.0
    elif 70 <= RS < 80:
        LG = 'B'
        GP = 2.0
    elif 60 <= RS < 70:
        LG = 'C'
        GP = 1.0
    else:
        LG = 'F'
        GP = 0.0
    return LG, GP

## View
# ウインドウを作り，ラベル，ボックスを作り，これらを配置する
root= tk.Tk()                          #ウィンドウの作成
root.geometry('600x600')
root.title('GPA (Grade Point Average) 計算器')

input_label = tk.Label(text = 'あなたのデータを入力してください') #ラベルを作る
input_label.pack()
subject_label= tk.Label(text = '履修科目名')    # ラベル
subject_label.pack()                   #  ラベルをパック
subject_box = tk.Entry()               # Entry ボックスを作る
subject_box.pack()                     # ボックスをパック

credit_label= tk.Label(text = '単位数')
credit_label.pack()
credit_box = tk.Entry()
credit_box.pack()

score_label= tk.Label(text = '点数')
score_label.pack()
score_box = tk.Entry()
score_box.pack()

calc_button = tk.Button()              # ボタンオブジェクト
calc_button['text'] = '計算ボタン'
calc_button.pack()

LG_label= tk.Label(text = 'LG')
LG_label.pack()
```

```
LG_box = tk.Entry()
LG_box.pack()

GP_label= tk.Label(text = 'GP')
GP_label.pack()
GP_box = tk.Entry()
GP_box.pack()

GPA_label=tk. Label(text = 'GPA')
GPA_label.pack()
GPA_box = tk.Entry()
GPA_box.pack()

fin_button = tk.Button()                    #  ボタンオブジェクト
fin_button['text'] = '終了ボタン'
fin_button.pack()

# 履修表の作成
tree = ttk.Treeview(root)                                  # 表（履修表）
tree['columns'] = (1, 2, 3, 4, 5, 6)                       # 列インデックスの作成
tree['show'] = 'headings'                                  # 表スタイルの設定
tree.column(1, width = 120)                                # 各列の設定
tree.column(2, width = 60)
tree.column(3, width = 50)
tree.column(4, width = 50)
tree.column(5, width = 50)
tree.column(6, width = 50)
tree.heading(1, text = '履修科目名', anchor='w')    # 各列のヘッダー設定
tree.heading(2, text = '単位数', anchor='w')
tree.heading(3, text = '点数', anchor='w')
tree.heading(4, text = 'LG', anchor='w')
tree.heading(5, text = 'GP', anchor='w')
tree.heading(6, text = 'GPA', anchor='w')
tree.place(x = 100, y = 350)                               # 表の位置設定

##  Controller
calc_button.bind('<Button-1>', GPA_handler)          # 関数オブジェクト，関数名のみ
fin_button.bind('<Button-1>', fin_handler)
root.mainloop()                                            # GUI の実行
```

実行結果

データ入力では，履修科目名は日本語でも可です。単位数と点数は半角の数字入力とします。点数が 100 点を超えると「計算ボタン」をクリックしても入力エラーとなり，点数を入れ直すように求め，履修表にもその科目の成績は表示されません。100 点以下の点数を入力して「計算ボタン」をクリックすると，正しく処理されて，LG，GP，GPA の値がボックスに入れられ，履修表に当該科目の 1 行の成績レコードが書き込まれます。１つの履修科目の処理が終えると，次の履修科目名，単位数，点数を上書きして「計算ボタン」をクリックします。すべての科目について入力を終え，正しい履修表が得られたことを確認します。GUI ウィンドウは必要に応じて画面コピーして HD に保存してください。「終了ボタン」をクリックすると，`root.destroy()` で `mainloop()` は 停止し，GUI ウィンドウが閉じて画面から消え，プログラムは終了します。

`Controller` では，ボタンの `bind()`メソッドを呼び出していますが，いずれもマウスの左シングルクリックによりイベントを生起させています。「終了ボタン」をマウスの左ダブルクリックとする場合には `'<Double-1>'` とします。

履修表は，表の形式が整えられ演算結果がより見やすい形で提示されています。これらのコードのメソッドが何をやっているかは，履修表と対応させてみると，大体のところが分かるかと思います。メソッドの詳しい説明は省きますが，今の段階ではそれでよいと考えます。履修表の配置の始まりは `place()`メソッドを用いて `(x, y)` 座標を入れてきちんと定めています。

課題 9.4

解答省略（各自に任せます）

9.2　GUIアプリの作成にPySimpleGUIを使う

課題9.5*

```
import PySimpleGUI as sg
## Model
def BMI_calc(values):
    height = float(values[0]) / 100
    weight = float(values[1])
    bmi = round(weight / (height*height), 1)
    return bmi

## View
layout = [
  [sg.Text('あなたのデータを入力してください')],
  [sg.Text('身長(cm) '), sg.Input()],
  [sg.Text('体重(kg) '), sg.Input()],
  [sg.Submit(button_text = '計算ボタン')],
  [sg.Submit(button_text = '終了ボタン')],
  [sg.Submit(button_text = 'BMI')]]
# ウィンドウオブジェクトの生成
window = sg.Window('BMI(Body Mass Index)計算器', layout)

## Controller
while True:
    event, values = window.read()      # 対話処理。イベントの読み込み
    if event == '計算ボタン':
        bmi = BMI_calc(values)
        sg.popup(bmi)
    if event == '終了ボタン' or event == sg.WIN_CLOSED:
        break
window.close()     # プログラムを終了する
```

このプログラムでは，計算結果はウィンドウ画面とは別の画面にポップアップして示されます。