プログラミング演習 Python 2023(Version 2023/10/12 (コラム編))

AUTHOR(S):

喜多, 一; 森村, 吉貴; 岡本, 雅子

CITATION:

喜多, 一 ...[et al]. プログラミング演習 Python 2023. 2023: 1-260

ISSUE DATE:

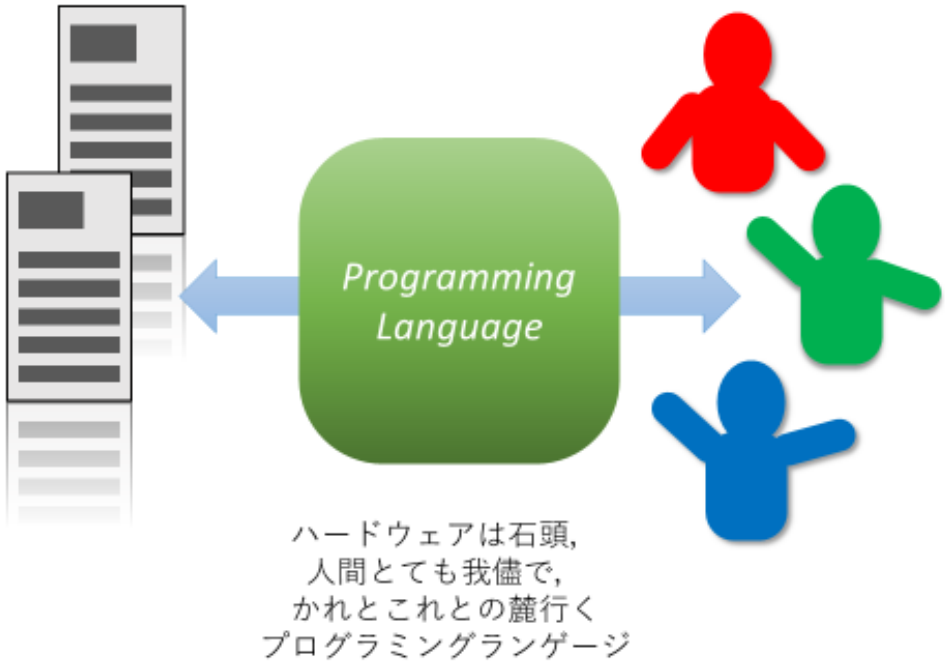
2023-10-17

URL:

http://hdl.handle.net/2433/285599

プログラミング演習 Python　2023

コラム編



京都大学　国際高等教育院　喜多　一
京都大学　情報環境機構　森村吉貴
京都大学　高等教育研究開発推進センター　岡本雅子

Version 2023/10/12

目次

目次 **2**
0. コラム 0 始まり **4**
0.1 Python は 0 ではじまる 4
0.2 1 始まりではいけないのか 4
0.3 結局は 5
1. コラム Float って？ **6**
1.1 浮動小数点数 6
1.2 ２進数で浮動小数点数を扱うことの嫌な点 7
参考文献 7
2. コラム ニュートン法 **8**
2.1 ニュートン法 8
2.2 ***n*** 乗根を求める 9
2.3 平均律 9
3. コラム 相対精度 **10**
3.1 数値の精度 10
3.2 絶対精度と相対精度 10
4. コラム 仮引数と実引数 **12**
5. コラム 変数のスコープ **14**
5.1 ローカル変数＝捨てる変数 15
5.2 Python での変数のスコープ 15
6. コラム 乱数 **16**
6.1 コンピュータと乱数 16
6.2 使いたい乱数 16
6.3 Random モジュールを使う 17
6.3.1 乱数の「種」を与える． 17
6.3.2 整数乱数をつくる 18
6.3.3 並べ替えを作る 18
6.3.4 実数乱数をつくる 18
6.4 一定の確率で実行する 19
6.5 Numpy での乱数生成 19
7. コラム 再帰 **21**
7.1 数式を処理すること 21
7.2 関係代名詞 21
7.3 再帰 21

8. コラム GUI 25
8.1 GUI とは 25
8.2 使いやすさの裏でのソフトウェア開発の大変さ 25
8.3 排除と包摂 26
9. コラム プログラムと日本語 —終わりそうで終わらない文字コードとの闘い 27
9.1 Python では UTF-8 27
9.2 ソースコード上の文字コード誤り 28
9.3 ¥記号に要注意 28
9.4 濁音にも注意 28
10. コラム 名前空間 29
10.1 名前の混乱 29
10.2 コンピュータの名前空間 29
11. コラム プログラムの文書化 31
11.1 プログラムはすぐに分からなくなる 31
11.2 プログラムの文書化 31
11.3 docstring 31
11.4 魔法の数字 32
参考文献 32
12. コラム 三角関数 33
12.1 ものづくりと三角関数 33
12.2 波としての三角関数 33
12.3 波形の違いを音として聞いてみる 34
参考文献 35
13. コラム 参照と複製 37
13.1 情報の参照と複製，その得失 37
13.2 Python での参照と複製 38
14. コラム 擬人化 40
15. コラム 逃げる 41

[1] 本編の表紙は「汽笛一声新橋を」で始まる鉄道唱歌東海道編の 48 番をもじったものです．新橋から神戸まで 66 番ある歌詞の中で京都については山科を含めると 9 番もあるのですが，創作されたのは 1900 年．残念ながら 1897 年に創設された京都帝国大学は登場しません．．

0. コラム 0 始まり

0.1 Python は 0 ではじまる

Python で数値の範囲を生成する際に使われる range 関数では range(5) とすると，0, 1, 2, 3, 4 と 0 から 4 までの 5 個の整数が生成されます．またリスト a = [0, 1, 2] の要素の先頭は a[0] と添え字に 0 を与えて操作します．C 言語など他のプログラミング言語の経験がなければ奇異に感じられるでしょう．

0.2 1 始まりではいけないのか

我々は日常生活でものを数えるときに 1 から数え始めます．数え終われば最後の数字が全体の個数になるので好都合です．

ではなぜ，0 から始めるのでしょうか．

1 番さんから 1 0 番さん用のデータの置き場所があって隣通しの番地がついているとします．データが最初に入っているのは 1 番さんの番地でこれを a とすると，2 番さん...10 番さんの番地は a+1, ..., a+9 となります．自分の番号と 1 番さんの番地に加える数値が 1 ずれるのです．もし 0 番さん〜9 番さんでいいなら，n 番さんの番地は a+n とすっきり計算できます．

また 10 まで数えることを考えます．1, 2, 3, ..., 10 となる訳ですが，最後の 10 は桁上がりが生じて 2 桁必要です．1 桁で扱える数は 0 を含めて 10 通りあるのに，1 から数えることで 0 を遊ばせる一方で，最後の 10 を表すのに 2 桁必要になってしまいます．

このような理由から計算を機械で支えているコンピュータにとっては 0 から使い始めることが好都合で，C 言語など多くのプログラミング言語で 0 始まりで使う慣習ができてきました．もっとも，プログラミング言語としては初期に作られ，今も科学技術計算で使われている fortran という言語では配列（Python のリストのように添え字で要素を操作できるデータの形式）の添え字は 1 始まりです．

数学の世界でも自然数の定義に 0 を含めるとする派と含めないとする派があるようです．

0.3 結局は

「郷に入らば郷に従う」という言葉があります．あきらめて慣れるのが近道ですが，数学のように 1 始まりの添え字を使う文化をもっている応用領域では注意して使いましょう．

1. コラム Float って？

1.1 浮動小数点数

Python で小数を扱うデータに変換する関数は float() です．プカプカ浮かぶという意味の語ですから，何だろうと思いますよね．実は，小数を扱うデータの型を浮動小数点数（floating point number）といい，その頭の語がこの関数名の由来です．ただ「浮動」する「小数点」と呼ばれても初心者はなんだかわからないかと思います．浮動小数点数と対立する概念は固定小数点数（fixed point number）です．

固定小数点数とは小数点の位置を固定した数値の表現です．例えば整数部 8 桁，小数点以下 4 桁を扱う，±〇〇〇〇〇〇〇〇．〇〇〇〇といった表現をとることです．

しかしながら，科学技術計算では絶対値の非常に小さい数，例えばプランク乗数（約 6.62×10^{-34} J s ）や非常に大きな数，例えばアボガドロ数（約 6.02×10^{23} mol^{-1} ）が用いられます．このような数値を表すには固定小数点数は辛く，表記例のように，「整数部を 1 桁として表記した数（仮数）」と「１０（基数）のべき乗（指数）」の積として表すことが効果的です．このような表現をコンピュータ内でも用いていて浮動小数点数と呼んでいます．すなわち，実際の小数点の位置がふわふわ揺れ動く数値という意味です．

コンピュータ（や関数電卓）の表示では１０のべき乗の指数を上付き文字で表現するのは煩雑なので，先のプランク乗数やアボガドロ数の数値部分は以下のように表記されます：

6.62E-34

6.02E23

計算機の能力が低かったころは，浮動小数点数を表すのに 4 バイトを用いることが行われていました（単精度浮動小数点数）．しかしながら，これは仮数部の有効数字が少なく，実用上の精度として問題もありました．そこで，実用上十分な精度のある表現として 8 バイトを用いることも多く，これは倍精度浮動小数点数(double precision floating point number) と呼ばれました．C 言語の浮動小数点数演算は倍精度での計算を標準として実装されましたが，その際のデータの型の名称で float は単精度の，double は倍精度の浮動小数点数を用いる意味で使われました．float

でもチンプンカンプンなのに，double って，と思う方もおられるでしょう．

Python では倍精度の浮動小数点数が一般に用いられていますが，文字列からの変換の関数は float と命名されています．

1.2 2 進数で浮動小数点数を扱うことの嫌な点

高速な計算を実現するために，コンピュータ内部では基数を 10 とする 10 進数ではなく，2 とする 2 進数が広く用いられています．整数を扱う際には 10 進数も 2 進数も本質的な違いはありませんが，小数では差異が出てしまいます．

10 進数の小数では 1/3 は 0.33333... となり有限の桁数では表せません，これは十進数の小数が各桁の大きさ 1/10, 1/100, 1/1000 ... のそれぞれ 0 ～9 倍の和として表そうとするためです．小学校で習ったときには違和感を持たれた方も少なくないと思いますが，もう慣れましたよね．

2 進数の小数では 1/2, 1/4, 1/8, 1/16 …が各桁の大きさとなり，その 0 倍か 1 倍の和で数値を表します．このため，10 進数を使う我々がしばしば用いる 0.1 が 2 進数では正確に表せません．小学校の時の違和感がコンピュータのおかげで私たちが日常的に用いる 10 進数の扱いで生じるのです．

実際，Python で 0.1 を 3 回加えたものが 0.3 と等しいかどうかを評価する以下の式について試してみると

```
0.1 + 0.1 + 0.1 == 0.3
```

その値は False となってしまいます[1].

科学や技術の計算では，十分な精度さえあればよい場合がほとんどで 0.1 が厳密な値をとっていることはあまり問題になりません[1]．しかしながら，お金を扱う会計計算では，金額としてかなりの桁数を扱う一方で，金利が 0.01 (1%) など，10 進数の小数で扱われることが多いです．このため，十進数の小数が正確に表せないことは避ける必要がでてきます．会計にコンピュータを用いるためには，コンピュータ内部でも 10 進数の計算をすることが求められます．Python では正確な 10 進数で演算する decimal というモジュールが用意されています．

参考文献

[1] 15. 浮動小数点演算，その問題と制限，Python チュートリアル
https://docs.python.jp/3/tutorial/floatingpoint.html

[1] 科学や技術で扱うのは実際の物理的な存在なので，実現可能な精度が十分に表せればいいのです．正確に長さ 1m の棒を作るとして，どの程度の精度を出せるかを想像してみてください．

2. コラム ニュートン法

この科目の最初の部分で平方根の近似値を求める計算を行いました．ある数 a の平方根 $\sqrt{a}$ の近似値 x_i について

$$x_{i+1} = \frac{x_i + a/x_i}{2}$$

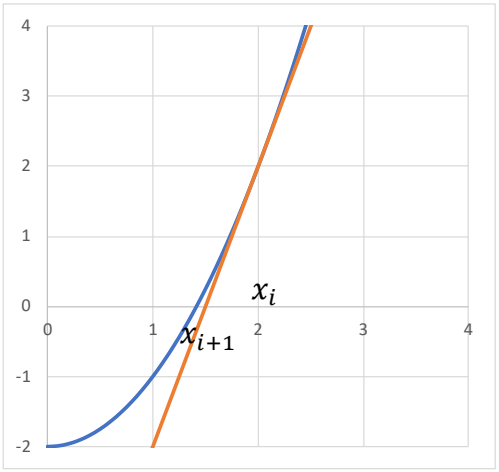
という式（漸化式）で更新してゆくというものです．式は単純で，収束も速いのですが逆数を記録する必要があるので手や電卓で計算したりするときには面倒です．

2.1 ニュートン法

この手法について，授業資料では直感的な説明を行いましたが，これは a の平方根を求める方程式

$$f(x) = x^2 - a = 0$$

の解をニュートン法（名称の由来はもちろん，アイザック・ニュートンです．ニュートン・ラフソン法とも呼ばれます）と呼ばれる手法で求めるアルゴリズムになっています．



ニュートン法では，方程式 $f(x) = 0$ の近似値 x_i について，$f(x)$ 上の点 $(x_i, f(x_i))$ で $f(x)$ の接線を引き，それと x 軸との交点を次の近似値とします．接線の方程式は

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

ですから $y = 0$ として x について解くと

$$x = x_i - \frac{f(x_i)}{f'(x_i)}$$

を得ます．これを x_{i+1} とするので

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

を得ます．平方根を求める場合には $f(x) = x^2 - a$ を当てはめると冒頭の漸化式を得ます．

$$x_{i+1} = x_i - \frac{x_i^2 - a}{2x_i} = \frac{x_i + a/x_i}{2}$$

2.2 n 乗根を求める

ちなみに n 乗根を求めたい場合は

$$f(x) = x^n - a = 0$$

として同じように漸化式を得ると

$$x_{i+1} = \frac{(n-1)x_i + a/x_i^{n-1}}{n}$$

を得ます．これは近似値 x_i とそれから計算したもう一つの近似値 a/x_i^{n-1} を $n-1:1$ に内分した点になっています．この式での計算のプログラミングは容易ですのでぜひ挑戦してみてください．

2.3 平均律

n 乗根と言われてもなじみがないかもしれませんが，音楽の音階では 12 乗根が使われます．音階で 1 オクターブ離れた音は周波数が 2 倍になります．私たちが不通に使っている平均律と呼ばれる音階では 1 オクターブを 12 の半音に割っていますが，半音異なる音程は周波数比が 2 の 12 乗根（約 1.059）だけ異なるように構成されています．和音をきれいに響かせるためには音程が整数比になっていることが望ましく，このように作られた音階を純正律というのですが，半音間の周波数比がばらついてしまい，転調などがしにくいことが問題になります．平均律は和音の多少の濁りを許して近似することで使いやすくした音階です．Python プログラムで音声データを作成することについては 12.3 節を参考にして下さい．

3. コラム 相対精度

3.1 数値の精度

理科では有効数字を気にして計算をしていたかと思います．自然科学では実験や観測で得られる計測値には誤差が含まれているためです．例えば，円の半径を計測して $r = 3.456\,\mathrm{m}$という値で表記されているということは小数点以下 3 桁の「6」という数値までが意味があるということを表しています．すなわちそれ以下の桁が四捨五入されていることから測定値rは

$$3.4555\,\mathrm{m} \leq r < 3.4565\,\mathrm{m}$$

を表し，不確かさの幅は $3.4565\,\mathrm{m} - 3.4555\,\mathrm{m} = 0.001\,\mathrm{m}$, すなわち 1mm ということになります．

同じ計測法で同程度の範囲を計れば 1mm 程度の誤差があるということになります．

これに対して，この円の外周の長さは $L = 2\pi r$で与えられますから計算すると

$$21.7115\mathrm{m} \leq L < 21.7178\,\mathrm{m}$$

となり，その誤差は $0.001 \times 2\pi \sim 0.00628$ となります．この場合，正確な定数 2π を掛けるので誤差の絶対値は変わりますが，L の値との比率はもとの 3.456:0.001 と変わりません．理科の計算で有効数字の桁数に留意して計算する理由はここにあります．計算を簡便に行うために正確に比率を吟味する代わりに，桁数で考えているのです．

3.2 絶対精度と相対精度

コンピュータを使った数値の計算では真の値に収束する数列などを使って近似計算します．この場合も，計算をどこで打ち切るかは必要な計算の精度から決まります．

例えば，真の値に収束する数列 $a_1, a_2, \cdots \to a$ を用いて a の近似値を計算することを考えます．仮に a_i の下限 a_i^L と上限 a_i^U も与えられているとすると，a_i の精度は

$$a_i^U - a_i^L$$

となります．この値が 0.001 より小さくなれば計算を終了させる，すなわち

$$a_i^U - a_i^L \leq 0.001$$

とすれば，計算の精度は 0.001 という絶対的な値になります．

しかしながら，例えば平方根の計算を考えると 10000 の平方根は 100 ですので，0.001 という絶対的な精度はかなり高いのに比べ 1/10000 の平方根は 1/100 = 0.01 ですので，0.001 の精度というのはかなり悪いことになります．

そこで，計算で得られる値に対して，一定の比率(例えば 0.001)の精度を要求するとすれば，そのための判定条件は

$$a_i^U - a_i^L \leq a \times 0.001$$

となり，両辺を a で割れば

$$\frac{a_i^U - a_i^L}{a} \leq 0.001$$

で判定することになります．ただし真の値 a は分かりませんので，実際に得られている計算値 a_i を用いて

$$\frac{a_i^U - a_i^L}{a_i} \leq 0.001$$

で判定することになります．このように相対精度で計算を終了させることは，理科の計算で行う有効数字の考え方を数値の近似計算に適用したものと言えます．

4. コラム 仮引数と実引数

Python で関数を呼ぶ例を考えてみましょう.

```
# ここは関数定義
def square(x):
 return x*x

# ここは呼び出し側
y = 2
y2 = square(y)
print(y2)
```

関数 square() は引数 x をとり，呼び出す側では変数 y を引数として関数 square() を呼んでいます．関数定義内の引数を特に「仮引数」，呼び出し側の引数を「実引数」と呼びます.

英語ではそれぞれ parameter (あるいは formal parameter), argument (あるいは actual parameter) と呼びます．初学者は呼び出す側と呼び出される側で引数の名前が異なることで混乱するかもしれません.

しかしながら，実世界でも同じようなことは使われています.

たとえば，食堂などで料理を注文することを考えてみましょう．以下のような会話はしばしば聞かれるものです：

> **ウェイター**：いらっしゃいませ，ご注文は？
> **客 B さん**：トンカツ定食
> **ウェイター**：かしこまりました．トンカツ定食ですね.
>
> ウェイター，厨房まで移動
>
> **ウェイター**：３番テーブルさん，トンカツ定食
> **料理人**：３番テーブル，トンカツ定食了解

実際のところ，食堂の従業員にとって，（よほどの馴染みのお客さんでもなけれ

ば)，お客さんの名前はどうでもいいことです．そこで客を特定するのに，「○番テーブル」という表現を用いています．これが食堂での仮引数と言えます．

仮引数を用いることで，食堂の側は業務の遂行が定型化でき，業務効率が上がります．客の側も食堂内で自分がどのように特定されているかはそれがうまく隠蔽されていれば問題ではないでしょう[1]．

プログラミング言語は，人が計算機を動かす命令をいかに容易に書けるか，ということで開発されてきました．その結果，現実社会での人の行動と類似したやり方が用いられるのだと考えられます．

[1] 「３番テーブル」という呼称で呼ばれるのは，それを聞かされるなら，あまりいい気がしない人もいるかもしれません．飛行機でビジネスクラスなどを利用すると，キャビンアテンダントに実名で呼ばれることがあります（運行管理上，チケットは実名で購入しています）．これはその客さんを大事にしているというサービスです．大学の授業はどうでしょうか？

5. コラム 変数のスコープ

プログラミング言語で使っている変数が「どの範囲まで見えているか」，を**「変数のスコープ」**といいます．

実世界で以下のようなことを考えてましょう．

資料などを作成する際には多くのメモなどがその途中で発生します．資料の作成の一部を部下に頼んだとします．

上司 A さん：B 君，すみませんが，この資料の図 3 の作成をお願いできますか．
部下 B 君：わかりました，A さんの作業台で作業すると資料など参照しやすいので，そこで作業していいですか．
A さん：いいですよ．使ってください．

B 君は，さまざまなメモをつくりながら，図 3 を完成させる．

B 君：A さん，図 3 ができました．じゃあ僕はもとの仕事にもどります．
A さん：ありがとう，たすかったよ．

B 君，そのまま立ち去る．

A さん：あいつ作業メモそのままにしてあるじゃないか．もとの資料と区別がつかん．困った．

さて，B 君はどうすればよかったでしょう．以下のようなことをすれば A さんを困らせずにすみます．

作業が終了したら自分の作ったメモは捨てる．

そのために，A さんの資料とメモを区別できるように，メモは専用の用紙をつかう．

あるいは，A さんの資料のない作業台にメモを置く

5.1 ローカル変数＝捨てる変数

プログラミング言語では関数などを呼び出すことで多くの仕事を下請けに出してしまいます．このときに，先の例での B 君のメモに相当する関数内で利用する変数はローカル変数として別に確保し，終了時に捨ててしまいます．関数の外側からはローカル変数は見えないようになっています．

5.2 Python での変数のスコープ

変数のスコープはプログラミング言語によって異なります．Python では

- 関数内の変数（ローカル変数）は外側から見えません．
- 関数内から外側で定義されたグローバル変数は読むことだけできます．
- 関数内からグローバル変数に書き込むためには global 文で宣言しないといけません．
- for 文の目標変数（例えば `for i in range(5):` の変数 `i` ）は for 文終了後も残ります．

6. コラム 乱数

6.1 コンピュータと乱数

コンピュータはプログラムで記述された動作を正確に実行します．コンピュータで数値の系列を生成することを考えると，プログラムと変数の値から，次に生成する系列の値は正確に予測できることになります．一方，乱数列とは「でたらめ」な数の列のことです．このため，外部からランダムな要素を持ち込むなどしない限り，本質的にはプログラムで乱数列は生成できません．

コンピュータで生成される乱数列は「ランダムっぽく見える」数列で正しくは「疑似乱数列」と呼ぶべきものです．生成される乱数列の良さはアルゴリズムによって定まりますが，さまざまなテストで，どのような意味で良い乱数列が生成されるのかを検討します．

コンピュータの応用では乱数はさまざまな所で使われます．皆さんにとって馴染みのあるものはゲームでしょうし，学術研究ではシミュレーションなどで多用します．乱数系列の発生では乱数系列を再現できるかどうかが問題になります．ゲームだと系列を再現できると面白さがなくなります．そこで，乱数系列の生成の初期値にコンピュータの時計を使ったりします．他方，学術研究では，同じ系列を再現できないと作業効率が落ちますし，他の方に追試してもらえません．そこで明示的に初期値を与えて乱数系列を作るようにします．

6.2 使いたい乱数

さまざまなプログラムを書いていて使いたくなる乱数としては以下のようなものがあります．

- 整数の乱数
  - 一様乱数，サイコロの目のように一定の範囲で同じ確率で生じる乱数
  - 特定の分布に従う乱数，例えば二項分布やポアソン分布に従う乱数
- 浮動小数点数の乱数
  - 一様乱数
  - 特定の分布に従う乱数，典型的には正規分布ですが，このほかにも様々な分布に従う乱数が必要になります．

- 並べ替え
  リストなどのランダムな並べ替え，抽選のように一定数をランダムになおかつ重複なく選びたい場合があります．

6.3 Random モジュールを使う

Python の乱数発生用モジュールとして random モジュールがあります．メルセンヌツイスターと呼ばれるよくテストされた手法で乱数を生成します．なお，セキュリティ用途にはより安全な secrets モジュールを使うことが推奨されています．

6.3.1 乱数の「種」を与える．

- random.seed()

乱数系列を発生させるための種を与える関数です．引数がなければ種としてシステムの時計を使います．整数型の引数を与えると，それを種として乱数系列を開始します． 例えば，次の例では seed() の呼び出しによって 0〜9 の乱数を発生する randrange(10) は異なる値を返します．

```
import random
random.seed()
random.randrange(10)
2
random.seed()
random.randrange(10)
3
```

一方，明示的に seed() に引数を与えると同じ乱数を返します．

```
import random
random.seed(1)
random.randrange(10)
2
random.seed(1)
random.randrange(10)
2
```

6.3.2 整数乱数をつくる

- random.randrange(stop)

引数は range() 関数と同じで，range() 関数で生成される範囲の整数乱数を生成します． 例えば random.randange(5) は 0 から 4 の乱数を生成します．

- random.randint(a, b)

a から b までの整数乱数を発生します．randrange() 関数と異なり，a も b も発生される乱数に含まれます．

6.3.3 並べ替えを作る

- ramdom.sample(population, k)

母集団 population を並べ替えて k 個を取り出します．母集団には range() 関数やリストが使えます．結果は並べ替えたリストが返ります．population として与えたリストは変更されません．

```
import random
a = list(range(6))
a
[0, 1, 2, 3, 4, 5]
b = random.sample(a, 6)
b
[4, 0, 2, 5, 1, 3]
a
[0, 1, 2, 3, 4, 5]
```

6.3.4 実数乱数をつくる

- random.random()

0 以上 1 未満の実数の乱数を作ります．

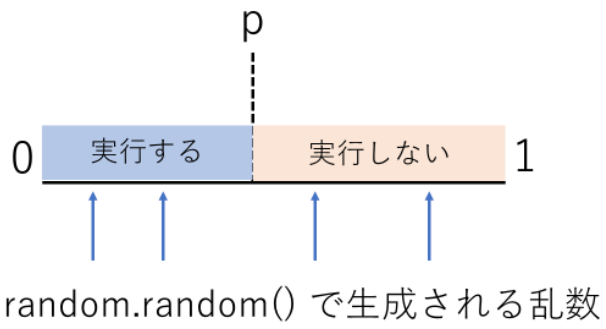
- random.gauss(mu, sigma)

平均 mu, 標準偏差 sigma の正規分布 (Normal Dsitribution, 数学者ガウス(Gauss)にちなんでガウス分布とも呼ばれます．)に従う乱数を生成します．

6.4 一定の確率で実行する

プログラムの中で，ある確率（p という変数の値としましょう）で何かを実行したい場合があります．その時には以下のようにプログラムします

```
if (p > random.random()):
    実行したいブロック
```



6.5 Numpy での乱数生成

高度な数値計算を支援するライブラリの Numpy では沢山の乱数を一度に発生させることが可能で，計算効率が上がります．多くの関数が定義されています．その一部は以下のようなものです． random モジュールと名称が異なることに注意してください．以下の例では numpy は慣習に従って np という名称で import しています．

- numpy.random.seed: 乱数の種を与える関数
- numpy.permutation 配列を与えるとそれを並べ替えた新しい配列を作ります．
- numpy.random.rand 連続一様分布に従う乱数を指定された個数生成します

  す ．引数を与えなければ，float 型の乱数を 1 つ返します．引数を与えるとその大きさの ndarray 型の乱数を返します．

```
import numpy as np
np.random.rand()
0.22963100707952988
np.random.rand(2,4)
array([[0.25512767, 0.92925799, 0.37337589, 0.76565738],
       [0.04984103, 0.77138769, 0.61715466, 0.0413605 ]])
```

- numpy.random.randint: 整数乱数を生成します．引数の与え方はやや複雑です以下の例を参考にしてください．

```
import numpy as np
// 0 から 5 未満の乱数
np.random.randint(5)
2
// 0（第一引数）から 4（第二引数）未満の乱数
np.random.randint(0,4)
3
// 0 から 4 未満の乱数を 2×4 の ndarray で生成
np.random.randint(0,4,(2,4))
array([[1, 1, 1, 2],
       [0, 3, 2, 3]])
// 5 未満の乱数を 2×4 の ndarray で生成，
// 第二引数を省略しているので size で指定
np.random.randint(5,size=(2,4))
array([[3, 2, 4, 4],
       [3, 1, 0, 3]])
```

- numpy.random.randn: 標準正規分布に従う乱数を生成します． 最後の n は正規分布の英語名 Normal Distribution の頭文字です．

7. コラム 再帰

7.1 数式を処理すること

私たちが日ごろ使う数式では計算の順序を制御するのに括弧（）を使います．

1×2 + 3×2 = (1+3)×2

と変形すると，２を掛けるという計算が一回で済みます．そのため，足し算を優先して行う必要があるので括弧 () の中に足し算を書きます．

括弧はその中にさまざまな数式を書くことができ，さらに内側に括弧があることを許します．われわれは，括弧が出てくるたびに，それまでの計算を「棚上げ」にして，括弧の中を計算します．さらに括弧が現れると再度，棚上げして括弧の中を計算します．

7.2 関係代名詞

英語の一つの文は名詞の主語と動詞を中心に目的語や補語で骨格が構成されます．名詞を修飾するのが形容詞ですが，形容する方法として関係代名詞があり，関係代名詞で修飾する部分には，再び主語や動詞が現れ，全体を「関係代名詞節」として捉えます．先ほどの数式と同じように関係代名詞節のなかに再び，主語（名詞）や動詞の構造が現れました．読解は難しくなりますが，この節の中の名詞をふたたび，関係代名詞節で修飾することも可能でしょう[1].

I read the book.

I read the book **that was referred in a book**.

I read the book **that was referred in the book** that was referred in another book.

7.3 再帰

先の数式の話でも関係代名詞の話でも，式や文の構造が入れ子になっているということが特徴で，その生成や解釈は棚上げにして内側で同じ手順を適用する，ということが求められます．プログラムで「部分」に対して同じ処理を適用する仕掛け

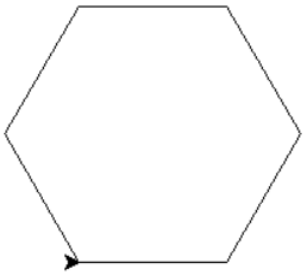
[1] このような文法構造は言語学の分野で，句構造文法と呼ばれます．チョムスキーによって提唱されたものですが，コンピュータでの構文解析にも活用されています．

を再帰 (recursion) と呼びます．よく用いられる例は階乗の計算です．N の階乗は N が 1 の時は 1, N が 1 より大きい（整数の）時は N! = N * (N-1)! と表せます．これをプログラムで書くと以下のようになります．

```
def fact(n):
    if n==1:
        return 1
    else:
        return n*fact(n-1)
```

今度は図形で考えてみましょう，以下は Turtle グラフィクスで 6 角形を描くプログラムです．

```
from turtle import *
for i in range(6):
    forward(100)
    left(60)
```



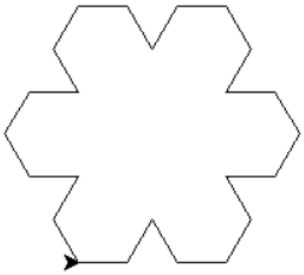
ここで forward(100) の部分を以下のように変形することを考えましょう．

- 1/3 の長さは直進
- 左へ 60 度回転
- 1/3 の長さで直進
- 右へ 120 度回転
- 1/3 の長さで直進
- 左へ 60 度回転
- 1/3 の長さで直進

このように迂回する関数を作って動かします．

```
from turtle import *
def detour(L):
    LL = L/3
    forward(LL)
    left(60)
    forward(LL)
    right(120)
    forward(LL)
    left(60)
    forward(LL)

for i in range(6):
    detour(100)
    left(60)
```

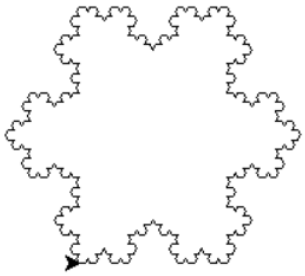


さらに，関数 detour の中で forward の代わりに自分自身である detour を呼ぶようにします．再帰の登場！です．ただし，長さが 10 以下ならそのまま forward で描画します．

```
def detour(L):
    if (L < 10):
        forward(L)
    else:
        LL = L/3
        detour(LL)
        left(60)
        detour(LL)
```

```
right(120)
detour(LL)
left(60)
detour(LL)
```

このような図形は一部がそれと相似な構造をもち，フラクタル図形と呼ばれます．自然界には雪の結晶のように，物理的な現象としてこのような構造が現れたり，木の枝のように生命の形づくりに現れたりします．



8. コラム GUI

8.1 GUI とは

GUI は graphical user interface の略で，現在のパーソナルコンピュータやスマートフォンで主に使われるユーザとコンピュータが対話するための環境です．これに対して conda prompt のような文字入力でコンピュータを操作する環境は CUI, character user interface と呼ばれます．IDLE の Python シェルは折衷的な存在であると言えます．

8.2 使いやすさの裏でのソフトウェア開発の大変さ

Tkinter は python で手軽に GUI 型のアプリケーションをプログラムを作成する環境ですが，実際のプログラミングに際しては，イベント駆動の考え方や，MVC アーキテクチャなどを理解したうえで，ウィジェットの配置，表示，動作など，細かく指定してゆくことが必要になります．実際にプログラムを書いてみるとプログラミングしなければならないことの多さに気づくでしょう．人に使ってもらうソフトウェアの開発にあたっては GUI の開発は全体の工数のかなりの部分を占めるようです．

しかも我々が直接プログラミングしているものの背景にはマウスやキーボードの入力を処理したり，ウィンドウの重なりを処理したりといった膨大な処理をウィンドウシステムが実現してくれており，さらに tkinter が Windows, macOS, Linux といった OS の違いを吸収する形で python で容易にプログラミングできる環境を提供してくれています．

スマートフォンなどタッチパネルを用いた GUI では，さまざまな指での操作（ジェスチャ）を解釈して動作を変えていますし，表示も画面のスクロールやメニューの出し入れなども人が使いやすいように細かくプログラムで調整されています．例えばスマートフォンなどではスクロールした画面が端に達すると単に止まるのではなく，跳ね返るようなアニメーションが行われます．もともと画面が跳ね返るようにできている訳ではなく，人が直感的に理解しやすいように，物理世界で起きそうなことをプログラムでわざわざ実現しているのです．

8.3 排除と包摂

GUI はコンピュータを直感的に操作することを可能し，その利用を拡大することに貢献しました．キーボードでの文字入力が十分に扱えない子供でも GUI ならある程度操作できます．

他方で，GUI によるコンピュータの利用から**排除 (exclude) されている人**が居ることも理解しておくことも重要です．視覚障害のある方には GUI 環境の利用は困難ですし，年配の方にはマウスボタンのダブルクリックを難しいと感じる方もおられます．指に障害のある方にはジェスチャを多用するスマートフォンの操作は難しいのではないでしょうか．

また，GUI は操作を人に教える，という点でも難しい側面があります．文字や言葉だけではなかなか操作法が伝わりません．コンピュータの利用法を書いた書籍や雑誌のページ数が多いのは画面を示さなければ操作法が伝えられないからですし，動画などの助けを借りなければ分からないことも多いかと思います．

「**排除 (exclude)**」の逆が「**包摂 (include)**」です．最近ではインクルーシブ・デザインと呼ばれるデザイン手法も注目されています．コンピュータは一方でソフトウェアの開発に費用がかかり，その回収も含めて商業主義的に健常者向けに開発されるものが多いのですが，同時に，適切にプログラミングすることで，より多くの人を包摂することも可能です．授業で紹介した MVC (model-view-control)アークテクチャでシステムが設計されていれば，コンピュータ担う本質的仕事を model として維持しながら利用者とのインターフェイスである view や control を利用者に合わせて構成しやすくなります．

9. コラム　プログラムと日本語
—終わりそうで終わらない文字コードとの闘い

9.1 Python では UTF-8

コンピュータの上で稼働する OS やプログラミング言語は英語圏で開発されたものが多く，アルファベットに比べ数多くの漢字を用いる日本語をコンピュータで用いることはどうしても後手に回ってしまいます．

それでも，世界の中で日本語は早くからコンピュータで利用できた言語です（！）．しかしながら，このことが日本語の文字をコンピュータ上で表す仕組みである「文字コード」の混乱を生じさせています．マイクロソフト社は早くから日本市場向けにソフトウェアを販売してきた会社ですが，Windows の前身のオペレーティングシステム MS-DOS では日本語用に Shift-JIS コードという文字コードを採用しました．その後，中国語や韓国語など多数の文字を用いる言語を含めて国際的に利用できる文字コードとして Unicode が開発され，Windows でも現在では内部ではこのコードが採用されています．ただ，日本での Windows では，従来のファイルとの互換性を維持するため**ファイル名やテキストファイルの文字コードに Shift-JIS コード**が用いられています．Apple 社の Mac も以前は日本向けに Shift-JIS コードを用いた OS を提供してきましたが，現在では Unicode を用いています．Linux でも近年では Unicode が標準になっています．

Python も Python 2 では文字コードの扱いに問題があったのですが，Python 3 では Unicode を採用していて， Unicode をコンピュータ内で扱う方法である文字符号化スキームとして utf-8 を用いています．Windows 環境では Python 自身がファイル名などに shift-JIS コードが使われているという情報を持っており，これにより文字コードの変換を行ってくれます．

よく，Python のソースコードの冒頭部分に

```
# -*- encoding: utf-8 -*-
```

というコメントを見かけます．これは，このソースコードが utf-8 で書かれていることを Python の処理プログラムに示す magic comment と呼ばれるものです．何も書かれていない場合も utf-8 であるとして処理されます．

また，このコメントの前に

```
#!/usr/bin/python
```

というものも見かけますが，これはシェバングとよばれ，Linux などの OS のシェルで Python プログラムを直接呼び出した時に起動するプログラムを示すものです．

9.2 ソースコード上の文字コード誤り

Python のソースコードで本来 ASCII コード（いわゆる半角の英数字）で書くべところを漢字コードの同じ文字（いわゆる全角文字）で書いてしまうと見つけにくいエラーになります．Python では字下げでブロックを表しますが，半角の空白文字の代わりに全角の空白文字を入れてしまうと見た目が変わらないために分かりにくいエラーなります．

9.3 ￥記号に要注意

ASCII コードのうち，バックスラッシュはこれまでの日本語のコードでは代わりに¥記号が割り当てられています．UTF-8 では別途，円記号用にコードが割り当てられているのですが，**Windows ではバックスラッシュ用のコード（5C）についても円記号のフォントで表示します**[1]．英語の資料などでバックスラッシュになっている箇所を円記号に置き換えて入力します．

他方で Mac では円記号を入力するとバッスラッシュ用のコードではなく，UTF-8 の円記号(A5)で入力され，本来，Python が求めているバックスラッシュ(5C)にはなりません．このためエラーになりますので，こちらはバックスラッシュで入力してください．

9.4 濁音にも注意

UTF-8 でのかな表記では濁音や半濁音は例えば「が」という文字を 1 文字で扱う方法と清音の「か」と濁音「 ゛ 」の 2 文字として表記し，合成して表示する方法の 2 とおりがあるようです．Mac で matplotlib を用いたりすると後者での文字表記に対応できず 2 文字として表示されてしまうことがあるようです．

グラフのプロットなどは見やすさの問題で済みますがファイル名にこの混乱があると大変分かりにくいエラーになったりするようです．

[1] Windows でも開発環境によっては，バックスラッシュを表示するフォントを選んだりしているようです．

10. コラム 名前空間

10.1 名前の混乱

名前空間というのは不思議な用語ですが namespace の訳で，名前が通用する論理的な空間というような意味です．

コンピュータについての名前空間の話をする前に，実生活で考えてみましょう．例えば田中一郎さん，という方がおられるとします．もし名前を使ってこの方を呼ぶとするなら，家庭内ではたぶん，「一郎」でしょう．田中という姓ではどなたのことかわからないからです．職場では「田中さん」と呼ばれることが多いでしょうが，同姓の方がおられたら「一郎」さんと呼ばれたり，「田中一郎さん」とフルネームだったりするでしょう．我々は自然とその場で一意になるように名前の呼び方を工夫しています．

電話番号はどうでしょうか？市内局番，市外局番，国際番号など電話を遠くからかけるために次第に長い番号をダイヤルしなければなりません．近くなら短い番号で特定できるように，階層的に番号をつける工夫がされているのです．

10.2 コンピュータの名前空間

コンピュータのプログラムでは多くのオブジェクト（変数とか関数とか）に名前をつけてアクセスします．ライブラリ（Python のモジュールなど）を使うと，多くの関数名などを利用しなければなりません．これらを特に階層化もせずに均一な空間でアクセスできるようにすると，同じ名称を別の意味で使ってしまう名前の衝突が生じます．そこで，名前の通用する範囲（名前空間）を制限することが必要になるのです．たとえば Python の数学ライブラリ math は

```
import math
```

として，この中で定義されている円周率は

```
math.pi
```

で呼び出します．少し面倒かもしれません．

そこで

```
from math import *
```

とすると，「math.」を付けずに利用できます．

```
sin(0.5*pi)
```

など，すっきり計算できて関数電卓代わりに使うのは便利です．ところが math では自然対数の底は

```
e
```

という変数で定義されています．うっかり e という変数に値を代入してしまうと，定義が崩れてしまいます．（残念ながら Python は定数の定義を保護する仕掛けがありません）．

このような混乱を招いてしまうことを「名前空間の汚染」と呼びます．名前空間を汚染しないためには安易な import は控えたほうがよいようです．

11. コラム プログラムの文書化

11.1 プログラムはすぐに分からなくなる

コンピュータのプログラムは書いた本人でさえ時間が経つとなぜそう書いたのかが分からなくなります．ましてや，複数人で継続的にプログラムを維持し，改善してゆくためには，プログラムについて説明した文書が必要になります．

しかしながらプログラムについての文書を作成することには多くの労力を要しますし，プログラムを書いている人にしか分からない事項も出てきます．プログラムについて文書化するもっともよい機会はプログラムを書いている時です．プログラマにとっても内容が分かっている時に書けばいいので，やる気も出ますし，作業効率も上がります．

11.2 プログラムの文書化

文書化の方法としてプログラムを読んで分かりやすくすることがありますが，それには２つの方法があります．

- 一つはプログラムそのものを読んで分かりやすいものにすることです．分かりやすい変数名や関数名をつけることなどが重要ですが，命名法や大文字，小文字などの文字使い，用語などを統一的につける**「コーディングの慣習」**も重要になります．Python のコーディング規約については PEP8 が知られています[2].
- もう一つはプログラムに注釈を入れることです．コーディング規約や注釈についての解説として文献が参考になります[3].

11.3 docstring

Python にはプログラムの文書化に関連して docstring という面白い仕掛けが導入されています．モジュールや関数，メッソドの冒頭にそれを説明する文字列(docstring)を定義しておきます．help() 関数は import 文などで読み込んだモジュール，関数などについて，この docstring を用いて説明文を作ってくれます．注釈と

異なるのは，プログラムとして定義された文字列ですので，他のプログラムから参照可能である，という点です．

11.4 魔法の数字

本書の本編で三目並べを扱った 15 章では以下のように手番や勝敗の状況について以下のように定数を定義して以下，プログラムの中ではこれらの定数を参照する形でプログラムを書きました．

```
OPEN = 0
FIRST = 1
SECOND = 2
DRAW = 3
```

もしプログラムの中で定数名ではなく 1 〜 3 の数値をそのまま書いてしまうとプログラムの意味を読み取ることが難しくなりますし，1 〜 3 などの数値が別の意味で使われている場所との区別がつかずプログラムの改修も難しくなります．

このようにプログラムに直接埋め込まれている数値のことは「**魔法の数字（マジックナンバー）**」と呼ばれ，プログラムを扱いにくくするので上記のように定数を定義してそれを参照するなどして避けることが望まれます．

参考文献

[2] PEP 8 -- Style Guide for Python Code, https://www.python.org/dev/peps/pep-0008/

[3] Al Sweigart 著，岡田佑一訳：きれいな Python プログラミング，マイナビ出版 (2022)

12. コラム 三角関数

12.1 ものづくりと三角関数

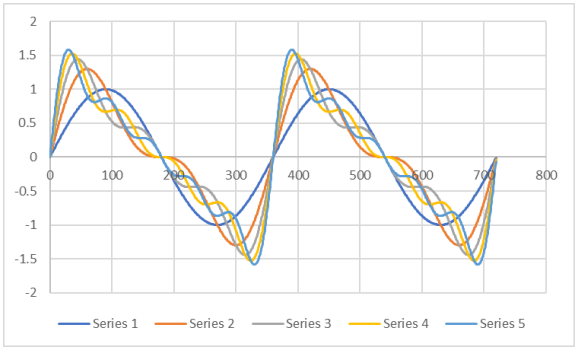
高等学校で三角関数を習ったときに，なんのために学ぶのかという疑問を持たれる方も少なくないと思います．この授業では tkinter のグラフィクス機能を使ってアナログの時計を作りました．時計の針を描くためには時刻を角度に読み直したうえで，針先の位置の X 座標，Y 座標が必要ですので三角関数は必須のものになります．三角関数を実用的に初めて使ったという方も少なくないかと思います．

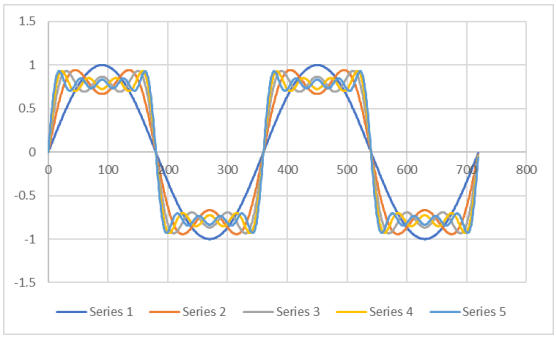
この例のように傾いたものの角度と水平，垂直方向の長さを関係づけるのが三角関数です．ですから，ものづくりには不可欠な関数になります．

12.2 波としての三角関数

もう一つの例として，周期関数を三角関数の和で表す例を示しました．のこぎり波や矩形波（方形波）はそれぞれ以下のように三角関数の和として近似されます[1]

- $f_1(x) = \frac{\sin(x)}{1} + \frac{\sin(2x)}{2} + \frac{\sin(3x)}{3} + \frac{\sin(4x)}{4} \ldots$
- $f_2(x) = \frac{\sin(x)}{1} + \frac{\sin(3x)}{3} + \frac{\sin(5x)}{5} + \frac{\sin(7x)}{7} \ldots$





このような特定の関数を近似する三角関数の和は「フーリエ級数」と呼ばれており，周期関数なら，その関数の周期の整数分の 1 の周期（整数倍の周波数）を持つ正弦関数 sin と余弦関数 cos の定数倍の和（級数）で近似できることが知られて

[1] 最大値，最小値が 1, -1 ののこぎり波，矩形波については，この式にそれぞれ係数 $2/\pi$, $4/\pi$ がかかります．

います．[1]フーリエ級数の各項の係数はフーリエ級数で表したい関数に，各項の三角関数を掛けて積分することで求められます．

12.3 波形の違いを音として聞いてみる

音程が同じでも楽器の種類によって音色が異なります．その理由の一つは，楽器の出す音には音程を表す周波数の正弦波（基本波，基音）に加え，その整数倍の周波数の正弦波（高調波，倍音）が含まれるためです．これは弦楽器の弦や管楽器の管が基本波だけでなく，高調波にも共振することによります．

ここでは，コンピュータで合成した波を実際に聞き比べてみましょう．

- sinwavw.wav は周波数 440Hz の正弦波
- harmonics-saw.wav はのこぎり波を第 7 高調波まででフーリエ級数で近似したもの
- harmonics_sq.wav は矩形波（方形波）を第 7 高調波まででフーリエ級数で近似したものです

これらのファイルは wav と呼ばれる形式の音声ファイルです．Python では wave というモジュールを使って wav 形式にファイルを読んだり，書いたりできます．上記のデータの生成に用いたプログラムが次の makesoundfile.py です（参考資料[4] を参考にしました）．Python で音のデータを扱うには文献[5]も参考になります．

プログラム 12-1 mksoundfile.py

行	ソースコード
1	import␣numpy␣as␣np
2	import␣wave
3	import␣struct
4	
5	#␣音声ファイル名
6	fname␣=␣'harmonics-sq.wav'
7	wf␣=␣wave.open(fname,␣'w')
8	#␣音声ファイルのパラメータ，1 チャンネル（モノラル），
9	#␣1 点の音声データ␣2byte␣(16bit)
10	#␣サンプリング周波数␣44.1␣kHz
11	ch␣=␣1
12	width␣=␣2

[1] 上記の例ではもとの関数が奇関数（グラフが原点に対称）なので，正弦関数（sin）しか現れません．また連続関数である三角関数の和で不連続な関数を近似しようと無理をしているので，関数の不連続点でその無理が現れてしまいます．

```
samplerate = 44100
wf.setnchannels(ch)
wf.setsampwidth(width)
wf.setframerate(samplerate)

# 10 秒間継続
time = 10
numsamples = time * samplerate

print( "チャンネル数 = ", ch)
print( "サンプル幅 (バイト数) = ", width)
print( "サンプリングレート (Hz) = ", samplerate)
print( "サンプル数 = ", numsamples)
print( "録音時間 = ", time)
print( "出力ファイル = ", fname)

# 信号データを作る (numpy の ndarray で)
freq = 440                    # 基本周の周波数 freq を 440 Hz にする
x=np.linspace(0, time, numsamples+1) # 0 ≦ t ≦ time を numsamples 等分
h1=np.sin(2*np.pi*freq*x)  # 基本波の正弦波
h2=np.sin(2*np.pi*2*freq*x)  # 第 2 高調波の正弦波
h3=np.sin(2*np.pi*3*freq*x)  # 第 3 高調波の正弦波
h4=np.sin(2*np.pi*4*freq*x)  # 第 4 高調波の正弦波
h5=np.sin(2*np.pi*5*freq*x)  # 第 5 高調波の正弦波
h6=np.sin(2*np.pi*6*freq*x)  # 第 6 高調波の正弦波
h7=np.sin(2*np.pi*7*freq*x)  # 第 7 高調波の正弦波

# 基本波と高調波に係数をかけて波を合成
y = h1 + 0*h2/2 + h3/3 + 0*h4/4 + h5/5 + 0*h6/6 + h7/7

y=np.rint(32767*y/max(abs(y)))  # [-32767,32767] の範囲に収める
y=y.astype(np.int16)            # 16 ビット整数に型変換する
y=y[0:numsamples]               # numsamples 個のデータに打ち切る

# ndarray から bytes オブジェクトに変換
data=struct.pack("h" * numsamples , *y)

# データを書き出す
wf.writeframes(data)
wf.close()
```

参考文献

[4] **桂田 祐史**：Python を使った WAVE ファイルの処理, http://nalab.mind.meiji.ac.jp/~mk/lecture/fourier-2018/python-sound/ (2018/12/3 アクセス)

[5] 青木直史：Python で始める音のプログラミング，オーム社 (2022)

13. コラム 参照と複製

プログラミング言語で利用者が注意すべきことはしばしば変数が扱う情報そのものではなく，実際の情報の所在への「参照」を持つことです．Python では変数はすべて，それが表す実体（オブジェクト）ではなく，それへの参照として統一的に扱われています．

13.1 情報の参照と複製，その得失

情報を参照することと複製することには得失があります．実世界で以下のようなことを考えてましょう．

A さん：B さん，先日の会議の資料をいただけませんか？
B さん：その棚にあるから，適当に使ってください．

数日後

B さん：A さん，先日の会議の資料を返していただけませんか？
A さん：適当に使ってくださいと言われたので，報告書に必要な部分を切り抜いてしまいました．残りをお返ししすればいいですか？

困りましたね．A さんと B さんが**同じ資料を共有**していたために，B さんが資料に手を加えた結果が A さんにも及んでしまうのです．この場合，**資料の複製**を作って A さんに渡すべきでした．

一方，A さんが資料の 1 部を閲覧したいだけなら，複製を作ることは手間ですし，A さんに B さんの作業の一部をお願いするなら，同じ資料を使ったほうが効果的です．

コンピュータ上では，情報を複製することは紙の資料などに比べれば容易ですが，それでも大量の情報を扱う場合には複製に要する時間のため実行効率が問題になってきます．害がないときは参照で同じ情報を見るようにし，独立に作業する場合に複製するのが効果的です．

13.2 Python での参照と複製

Python では変数はすべて，実際のデータへの参照として統一的に扱われます．他方で，データについては数値や文字列は内容の書き換えを許さないイミュータブルなデータとして扱われるため他のプログラミング言語に慣れた方でも比較的自然にプログラムを書くことができます．ただし，リストなどは内容の書き換えを許すミュータブルなデータですので以下の点の注意が必要です．

- 関数呼び出しなどで引数として与えたデータの書き換え．
  以下の例では変数 a は整数型のデータ，b はリストが代入されています．関数 f() の引数として a, b を与えた場合，f() の実行後は a の値は書き換わっていませんが，b の 0 番目の要素は書き換えられています．

```
a = 1
b = [1,2,3]
def f(x, y):
    x = 0
    y[0] = 0
a, b
(1, [1, 2, 3])
f(a, b)
a, b
(1, [0, 2, 3])
```

- グローバル変数の書き換え．以下の例ではグローバル変数 a はリストですが，関数内では global 宣言していません．しかしながら a の要素は書き換え可能です．

```
a = [1,2,3]
def f():
    a[0] = 0
a
[1, 2, 3]
f()
a
[0, 2, 3]
```

- リストなどのデータの複製の作成．リストの複製を使いたいときには以下のように明示的にそれを生成します．

```
a = [0, 1, 2]
b = a.copy()
a[0] = 1
a
[1, 1, 2]
b
[0, 1, 2]
```

14. コラム　擬人化

コンピュータのプログラミングでは -er などの接尾辞がついた用語によく出会います．これは自動的に動いてくれるコンピュータに何かを頼むので擬人的な表現が適しているからです．クラスの説明の中で「コンストラクタ (constructor)」が出てきました．

この教科書では複数のタートルを扱うことを体験していただいたうえで，クラスという章の中でオブジェクト指向について解説しています．その中では**オブジェクトはロボットのようなもの**だと説明しました．オブジェクト指向の教科書の中にはオブジェクト＝もの，として説明しているものあるのですが，オブジェクトがメソッドの呼び出しで能動的に動くことをイメージしにくいと思っています．実際には擬人化した名称をつけてオブジェクトを作ることが少なくありません．

コンピュータ(computer)という語も機械式の計算機が出現するまでは，計算を担当する人（計算手）を意味していたようです．自動化により，それまで人間が行っていた仕事が機械に置き換えられてきました．

コンピュータが安くなったことで，すべてのものにコンピュータが組み込まれネットワークにつながる IoT (Internet of Things)が注目されています．今までは人の仕事を機械に置き換えてきましたが，これからは，「物」が人にように動いてくれる，と考えたほうが分かりやすいのかもしれません．擬人化して妄想することが現実になるかもしれないのです．相変わらず，紙の書類や書籍はたくさん使われるのですが，私のように片付けるのが苦手な人間にとっては勝手に片付いてくれる書類や書籍が欲しいところです．さて，あなたは何を擬人化する妄想を抱きますか？

15. コラム　逃げる

プログラムのソースコードなどはテキストファイルとして文字の並びとして書きますが，特定の記号を特別な意味で使うことが少なくありません．例えば，文字列定数は " で囲って

```
"文字列"
```

などと表記しますが，それでは「"」を文字列に含めたいときにはどうすればいいのだ，ということが生じます．すなわち，「命令としての解釈」から「逃げる」手段が必要になります．Python では文字列を囲むのに，シングルクォートとダブルクォートのどちらでも使えるので，少し便利です．例えば "文字列" 全体を文字列として扱いたいときには

```
'"文字列"'
```

と書けばいいからです．

文字列の中に改行などの記号を挿入したいときには ¥n と書きます．¥で始まる書き方を「エスケープシーケンス」と呼びます．まさに，逃げる，ですね．それでは「¥」という文字はどう書くかというと 2 文字続けて「¥¥」と書くようになっています[1]．

数値などを文字列に変換する書式（format メソッドで解釈するもの）では「{}」が変換の書式として特殊な意味をもちます．それでは「{」や「}」そのものを文字列として扱いたいときにはどうするのだ，ということが気になります．この場合は「{{」のように 2 つ続けて書くことになっています．

プログラミングを学ぶときに文字列表記などで「解釈」から「逃げ」たいことは，割によく出会います．逃げ方を確認しておくことは細かなことですが気にしておくといいでしょう．非常口の確認はプログラムの世界でも必要です．

[1] プログラムで「他のプログラム」を生成したいこともしばしばあります．生成後のプログラムの中でエスケープシーケンスが正しく出力されるように，生成するプログラムでエスケープシーケンスが解釈されないようにしないといけません．