Learning PYTHON

Launch 20220110、Modify 20220111～20231125 recent review

Word 編集に「不慣れ」で本文コードのインデントに乱れがございます
コーディング時に修正を…

INDEX

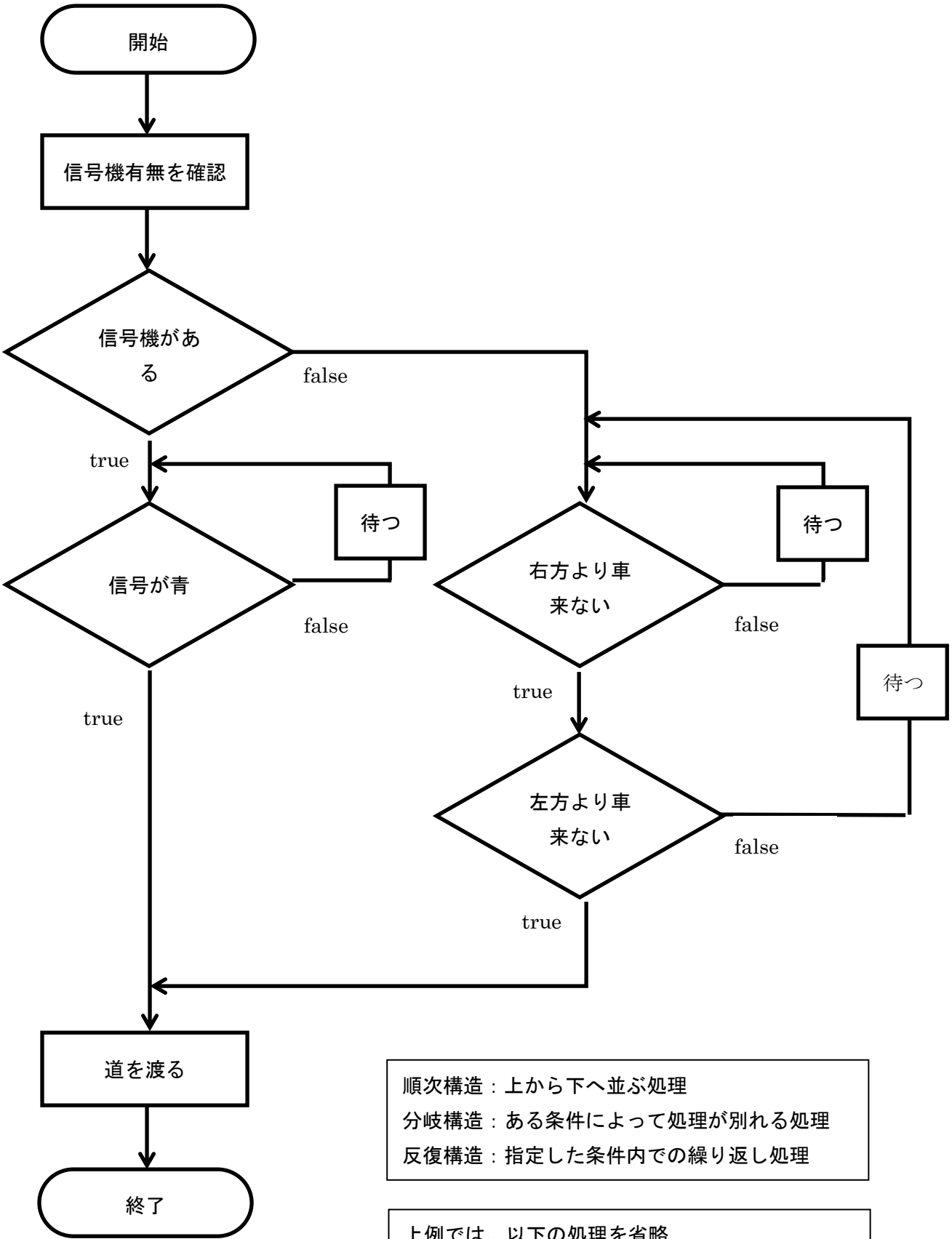
1. 過去の実践 VBA、JavaScript、Pov-Ray　2. 現在 Python とその根拠

3. 作法と課題（アプリケーション開発～人間の思考をマシーンに実装する体験）

アルゴリズム プログラミング

道路を渡る行動の流れ FCD

アルゴリズム（問題解決の算法）開始



順次構造：上から下へ並ぶ処理
分岐構造：ある条件によって処理が別れる処理
反復構造：指定した条件内での繰り返し処理

上例では、以下の処理を省略
信号機を確認する
道路の右（左）方向を確認する

INDEX

アルゴリズム(問題解決の算法)終了

問題解決の解法(算法)➡アルゴリズム

問題解決の解法(アルゴリズム)をPCに実装 検査修正すること➡**プログラミング**

プログラミングにおいて目的に合った言語を書くこと➡**コーディング**

1. 過去の教科情報実践:18 年前～近年➡下記(1)～(4)　目的が限定的

(1)Microsoft 製品　VBA 資料 strnun mountain view(当サイト)より

http://strnun.fool.jp/pov-ray_strnun/VBAintro.pdf

(2)WEB(HTML)連携　Javascript 資料 strnun mountain view(当サイト)より 2 点

http://strnun.fool.jp/pov-ray_strnun/joho_strnun_js1.html

http://strnun.fool.jp/pov-ray_strnun/js-2.pdf

(3)3DCG　Pov-Ray 資料 strnun mountain view(当サイト)より

http://strnun.fool.jp/pov-ray_strnun/3D%20programming%20with%20ray%20tracing%20technique%20part-1.pdf

(4)Flash　ActionScript➡フラッシュ自体が WEB から「消えている」

INDEX

2. 現在の教科情報の主流　Python

(0)根拠 機械学習 AI 連携～Society 5.0～（労働）生産性向上、モジュール ライブラリ活用

統計解析 デザイン等多目的、2025 年以降大学共通テスト予想問題との近似性　Python

(1)環境設定

①Web プラットホーム（Google クラウド内　先ず学校で設定された Google アカウントでログイン）

注：個人 Google アカウント　個人利用の永続性にメリット、協働学習にデメリット

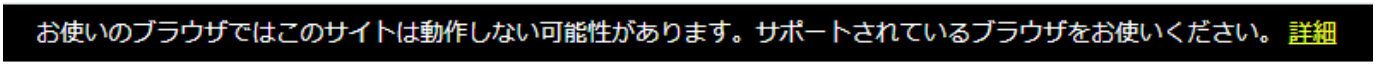
学校 Google アカウント　協働学習にメリット、個人利用の永続性にデメリット（卒業時に消失）

② Colaboratory ブラウザから Python を記述、実行できるサービス（下記 URL）

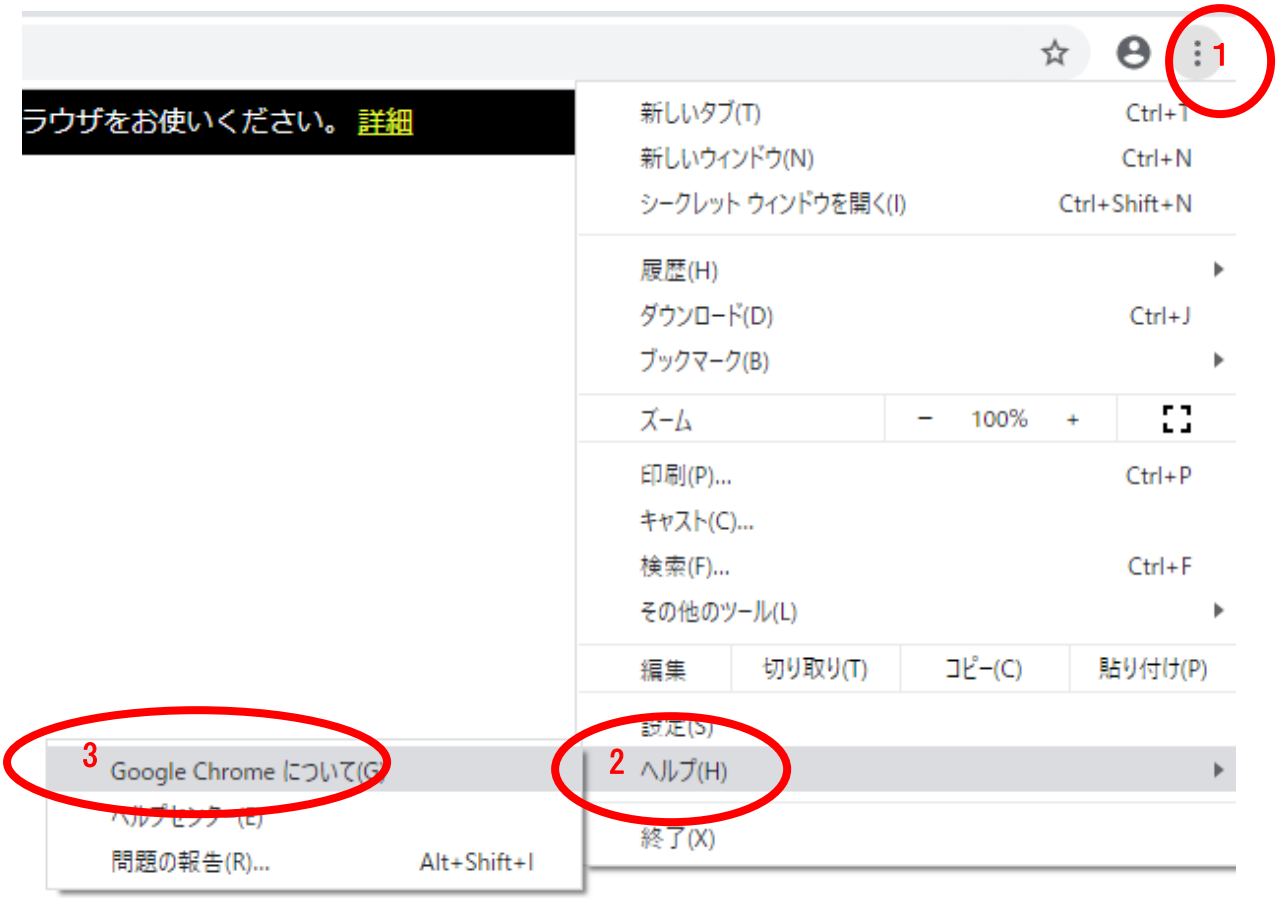
環境構築が不要、GPU への無料アクセス、簡単に共有

https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja

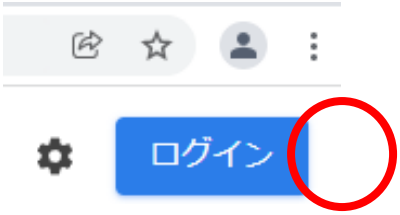
注：ここで下のメッセージが登場した場合



ブラウザ Chrome の以下 1～3 をクリックすると、次図になり 4 再起動ボタンをクリックして準備完了



③ログイン



④ファイル➡ノートブック新規作成

⑤以下に実装コード書き込み、実行する➡自動保存されるのでＲ。

コラム：資料 Artificial Intelligence 参考リンク例：文部科学省様、NTT 東日本様、YouTube 様

https://www.mext.go.jp/kids/find/kagaku/mext_0008.html AI ってなに？

https://business.ntt-east.co.jp/content/cloudsolution/column-166.html#section-01 機械学習とは？

https://www.youtube.com/watch?v=fU0fmF2N2qk&t=65s マツコロイド、黒柳徹子アンドロイドと初対談

https://www.youtube.com/watch?v=LvmGeJl5yjA&t=23s トットの部屋 第1回 ゲスト 黒柳徹子さん【totto】

https://www.youtube.com/watch?v=FHdLH2yESzE AI ソフィア VS AI ハン

https://www.youtube.com/watch?v=07j9M42PmRQ AI ソフィア

https://www.youtube.com/watch?v=WbpYLvlPoQ8&list=TLPQMjAwMTIwMjJ-KTwabrXhGQ&index=3 AI エリカ

確認事項：統計分析の手法 Excel（PC+アプリ）➡ プログラミング+ネット…ブラウザだけでよい

データベース　Access（PC+アプリ）➡ SQL ＋ ネット　…ブラウザだけでよい

PCアプリの時代＝過去のIT　➡ WEB活用　ブラウザ活用DX＝Society5.0 の IT

3. コーディング作法

(1) 数値型(整数 int,小数 float)、演算子

```
print(3)        # 値
print(5+2)      # 加算
print(5-2)      # 減算
print(5*2)      # 乗法
print(5/2)      # 除法
print(5//2)     # 商
print(5%2)      # 剰余
print(5**2)     # 累乗
x=5
x+3             # print 省略
x=10/3          # x に 10/3 を代入
print(x)        # x 表示
print(int(x))   # 整数型
print(x)        # x 表示
print(float(x)) # 小数型
print(x)        # x 表示
print(float(3e-2))  # 小数型 3*10^-2
```

(2) テキスト型(str)　string　　# 文字を扱う

```
print('love')          # 値
print('love'+'&'+'Peace')    # 値
'today'             # print 省略
x='everyday'
y='every'+'¥n'+'day'       # 改行
print(x)            # 変数 x
print(y)            # 改行 y

t='world wide web'        # 文字列定義
print(t)            # 文字表示
print(len(t))           # 文字数表示
```

コラム:改行

```
print("day")
print(5)
```

```
day
5
```

```
print("day",end = "")
print(5)
```

```
day5
```

(3) ブール型(bool)　　# true、false の二択

```
print (1>0.6)
print (1<0.6)
print (1==0.6)　　# 1 と 0.6 は等しい(等号:==)
print (1>=0.6)
print(8!=2)　　# 1 と 0.6 は等しくない(不等号:!=)
```

演算子比較	比較演算子	数学記号
左辺が右辺と同値	＝＝	＝
左辺が右辺と違う	！＝	≠
左辺が右辺未満(小なり)	＜	＜
左辺が右辺以下(小なりイコール)	＜＝	≦
左辺が右辺より大(大なり)	＞	＞
左辺が右辺以上(大なりイコール)	＞＝	≧
代入	＝	

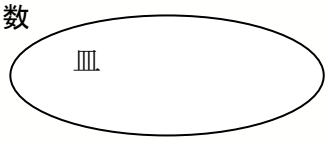
(4) 型変換

①python では値を入力した時点で自動的に「型」を振り分けている　(他の言語では変数定義で型を指定)

```
x=10　　# 10 を数値として扱う
x='10'　　# 10 を文字として扱う
print(int(x)+20)　　# 文字 10 を整数として 10+20 を表記
print(x+4)　　# 文字型+整数型は禁則➡エラー
```

(5) 変数

変数



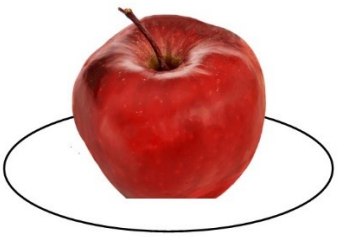
値

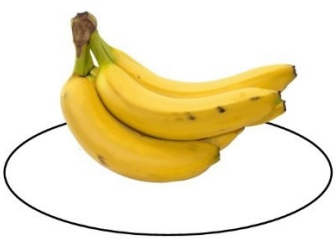

値:リンゴ

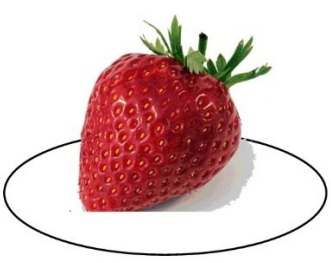

値:バナナ


値:イチゴ

変数に値を代入







皿=リンゴ　　皿=バナナ　　皿=イチゴ

この場合、演算子は = である。同値の場合は == である。

①コーディング

```
a=7        # 変数 a=7 とする
print(a)     # 変数 a=7 を表記
```

②比較してみよう

```
a=100       # 変数 a=100 とする
b=10        # 変数 b=10 とする
c=1         # 変数 c=1 とする
a*3+b*5+c*7    # print 省略すると
a*5+b*2+c*3    # どうだった？
```

```
a=100
b=10
c=1
x=a*3+b*5+c*7      # 変数 x を定義
y= a*5+b*2+c*3     # 変数 y を定義
print(x)           # print を入れると
print(y)           # どうだった？
```

課題 底辺 b、高さ h の三角形の面積を s とし、底辺 6、高さ 4 の場合の面積を求めるコード。

```
b=6
h=4
s=b*h*0.5
print(s)
```

(6) input()関数：キーボードを使った文字列データの入力可（プロンプト）

```
input("what's your name?")       # 名を問う
input("How old are you?")        # 年齢を問う
x=input("what's your name?")     # 入力値を変数 x とする
print(x)                 # どうだった？
y=input("How old are you?")      # 入力値を変数 y とする
print(y+1)               # どうだった？ 学び：input で獲得した値は文字型
```

例題　使命と年齢を回答させて還暦までの年数を返すプログラム

```
x=input("what's your name?")
y=input("How old are you?")
print(x,60-int(y),' years left until the 60th birthday')
```

氏名 還暦までの年数

課題 本体価格を回答させて、10%の税込み価格を提示するプログラム

```
price=input('How much?')                  # 本体価格を問う prompt
print('Payment is',float(price)*1.1,'yen including tax') # 税込み価格を提示
```

(7) 条件分岐

①成人ですか？

```
age = 25              # 変数 age に 25 を代入
if age < 18 :         # 変数 age が 18 未満であるか？
  print("minor")      # true(yes)ならば、minor を表記
else:                 # 否定ならば
  print("adult")      # false(no)ならば、adult を表記
```

②3の倍数ですか？

```
n = 60
if n%3 ==0:              # 最後に半角コロン: 改行後インデント（字下げ）必須
  print('Multiple of 3')  #  n を 3 で割った剰余が 0 ならば「Multiple of 3」
else:                    # そうでなければ
  print(n)               # 数値 n
```

③3の倍数ですか？ 5の倍数ですか？

```
n=30
if n%3==0 and n%5==0 :      # n を 3 で割った剰余が 0、かつ 5 で割った剰余が 0 ならば
   print('Multiple of 15')  #「Multiple of 15」
elif n%3==0:                # そうでないとき n を 3 で割った剰余が 0 ならば
   print("Multiple of 3")   #「Multiple of 3」
elif n%5:==0:               # そうでないとき n を 5 で割った剰余が 0 ならば
   print(' Multiple of 5')  #「Multiple of 5」
else:                       # そうでなければ
   print(n)                 # 数値 n
```

課題 名前 n、身長 h、体重 w とする。bmi=w/(h*h)を算出し以下の判定を返すプログラム

bmi25 以上なら over weight、18.5<未満なら under weight、18.5 以上 25 未満なら healthy weight

表示例：n さん、貴方の BMI は bmi、healthy weight

```
n = input('what is your name?')                    # name を問う
h = input('How tall are you in meters?')           # height（小数も有だね）を問う
w = input('How much is your weight in kilograms?') # weight（小数も有だね）を問う
bmi=float(w)/float(h)**2                           # bmi 計算式（小数の計算）
if bmi>=25:                                         # 25 以上ならば
  print(n,',BMI=',bmi,'over weight')                # over weight
elif bmi <18.5:                                    # そうでなければ、18.5 未満ならば
  print(n,',BMI=',bmi,'under weight')               # under weight
else:          # そうでなければ、
  print(n,',BMI=',bmi,'healthy weight')             # healthy weight
```

参考：Body mass index (BMI)

(8) 反復

① for 文　for 変数 in range(回数) ～ 処理

```
for n in range(10):        # 変数 n を 10 回繰り返して➡最後に半角コロン:
  print(n)                 # 改行後インデント(字下げ)必須　n を表示せよ

for n in range(10):
  print(n+1)               # n+1 を表示せよ

for n in range(0,10,3):    # 変数 n を 0～10 未満の間で 3 毎に繰り返し
  print(n)
```

range の扱い　range(start, stop[, step])

start ～ stop 未満で step ごとに連続した数値を返す。

start 省略→0、step 省略→1

```
for n in range(0,10,2):      # 変数 n を 0～10 の範囲で 2 つおきに繰り返せ
```

② 反復 for に分岐 if の入れ子

例題 ①の範囲で偶数のみ表記させてください

```
for n in range(0,10):      # 変数 n を 0～10 未満の範囲で繰り返せ
  if n%2==0:               # 変数 n を 2 で割った余りが 0 ならば
    print(n)               # 変数 n を表示せよ
```

例題 ①の範囲で奇数のみ表記させてください

```
for n in range(1,10):      # 変数 n を 1～10 未満の範囲で繰り返せ
  if n%2==1:               # 変数 n を 2 で割った余りが 1 ならば
    print(n)               # 変数 n を表示せよ
```

例題 掛け算九九　for の「入れ子」 m × n = m * n

文字表記　数値計算

```
for m in range(9):              # 変数mは 0～8(9 回)繰返し
  for n in range(9):            # 変数nは 0～8(9 回)繰返し
    print((m+1),'×',(n+1),'=',(m+1)*(n+1))   # 右辺は(1～9)* (1～9)の計算

for m in range(1,10):           # 変数mは 1～9(9 回)繰返し
  for n in range(1,10):         # 変数nは 1～9(9 回)繰返し
    print(m,'×',n,'=',m*n)      # 右辺は(1～9)* (1～9)の計算
```

課題 x年y組z番を以下の条件で全て表示せよ。

1＜＝x＜＝3、1＜＝y＜＝8、1＜＝z＜＝45

③ while 文

```
n=0              # 変数n初期値 0
while n<10:        # 変数nを 10 未満で繰り返し
  print(n)         # 変数nを表示
  n=n+1            # 変数nに1を加えて処理
n=5              # 変数n初期値 5
k=6              # 公差 k=6 とする
while n<=20:       # 変数nを 20 未満で繰り返し
 print(n)          # 変数nを表示
 n=n+k           # 変数nに k=6 を加えて処理
```

④ 反復 while の入れ子

例題 掛け算九九　while の「入れ子」

```
n=1                          # 変数n初期値 1
while n<=9:                    # 変数nを 9 以下で繰り返し
m=1              # 変数m初期値 1
while m<=9:          # 変数mを 9 以下で繰り返し
  print(n,'*',m,'=',m*n)     # m × n = m * n
  m=m+1            # 変数mに1を加えて処理
  n=n+1          # 変数nに1を加えて処理
```

⑤ 反復 while の強制終了

```
x = 0                        # 変数 x 初期値 0
while x <= 20 :                # 変数 x を 20 以下で繰り返し
  print(x)                     # 変数 x を表示
  x += 3             # 変数xに 3 を加えて処理
  if x > 12:           # 変数xが 12 を超えた場合
    break;             # 強制終了
print("fin")           # fin を表示
```

⑥ バリエーション

```
total=0          # total 初期値 0
for i in range(1,10):    # 変数 i を 1～10 未満の範囲で繰り返せ
 total += i        # 値を順に加算せよ total
print(total)

total=0          # total 初期値 0
for i in range(1,10,2):   # 変数 i を 1～10 未満の範囲で 1 つ置きに繰り返せ
 total += i        # 値を順に加算せよ total
print(total)
```

(9) 乱数

①賽の目

```
import random                  # 乱数を取り入れる
dice=[1,2,3,4,5,6]             # dice 配列 1～6
print(random.choice(dice))  # 乱数で dice 配列から値を選択
```

②運勢　通称「おみくじアプリ」

```
import random                  # 乱数を取り入れる
fortune=['excellent','luck','good','unknown','bad luck','terrible']
print(random.choice(fortune))
```

課題　乱数を使う「PC VS You じゃんけんアプリ」➡クラウド対あなたのじゃんけんゲームを作成せよ。

考え方　G Cloud 側はグー:0、チョキ:1、パー:2 として乱数を発生させ、判定は下表とする。

You⇓　PC⇒	グー　:0	チョキ:1	パー　:2
グー　:0	Draw	You Win	You Lose
チョキ:1	You Lose	Draw	You Win
パー　:2	You Win	You Lose	Draw

ヒント　① グー:0　チョキ:1　パー:2

② あなたの手:(6) input()関数 (input('>>')) を使う

③ 最後に勝敗の判定

実装

```
import random                          #  乱数をインポート
list=[‘stone’,’ scissors’,’ paper’]    # 配列グー・チョキ・パー
x=(random.randint(0,2))                #  乱数 0,1,2 を変数 x とする
if x==0:                               # x が 0 ならば
  print('PC',list[0])                  #  PC は石
elif x==1:                             # x が 1 ならば
  print('PC',list[1])                  # PC はハサミ
else:                                  # x が 2 ならば
  print('PC',list[2])                 # PC は紙
y=int(input('What do you choose?'))    # You の選択を問う
if y==0:                               # y が 0 ならば
  print('you',list[0])                 # you は石
elif y==1:                             # y が 1 ならば
  print('you',list[1])                 # you はハサミ
elif y==2:                             # y が 2 ならば
  print('you',list[2]))                # you は紙
if y==x:                               #  x と y が同値なら
  print('draw')                        #  Draw 表記
elif (y==0 and x==1) or (y==1 and x==2) or (y==2 and x==0):  # 以降は上表から判定表記
  print('you win')
elif (y==0 and x==2) or (y==1 and x==0) or (y==2 and x==1):
  print('you lose')
else:                                  # 以外ならば
  print('you foul')                    # you は反則
```

(10) 描画ツール

(10-1) 図形「亀」環境設定

① コード　Colaboratory 固有のツールをインストール

次回に別ノートでコーディングする場合、下記 2 行と③1 行目をセットでコピーをすると速い。

```
!pip3 install ColabTurtle
from ColabTurtle.Turtle import *
```

②結果

```
Requirement already satisfied: ColabTurtle in /usr/local/lib/python3.7/dist-packages (2.1.0)
```

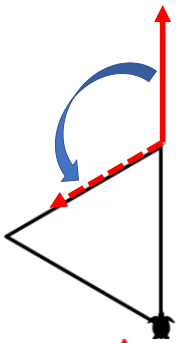
注釈：Google との通信を再開するときに、①②の手順を実行する

③動作確認　上から順に読み込み実行する　順次構造

```
initializeTurtle()      # 描画エリアの初期化
pencolor('green')       # 緑色
left(90)                # 左 90 度を向け　そちらが正面
forward(200)            # 200 進め
pencolor('blue')        # 青色
forward(200)            # 200 進め
initializeTurtle()      # 描画エリアの初期化
goto(100,200)           # 座標(100,200)へ移動
goto(0,0)               # 座標(0,0)へ移動　ディスプレイの左上が原点(0,0)座標表記
turtle.reset()          # 初期化
initializeTurtle()
penup()                 # ペンを放せ
goto(100,200)           # 座標(100,200)
pendown()               # ペンを下ろせ
goto(0,0)               # 座標(0,0)へ
```

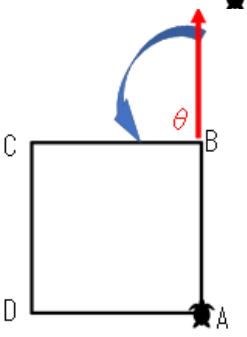
④例題：正三角形描画コーディング

```
initializeTurtle()      # 描画エリアの初期化
for i in range(3):      # 変数 i を 3 回繰り返す
    forward(200)        # 200 進む
    left(120)           # 左に 120 度回転
```

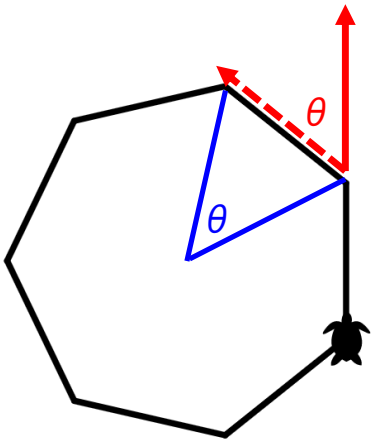


⑤例題：正方形描画CODING

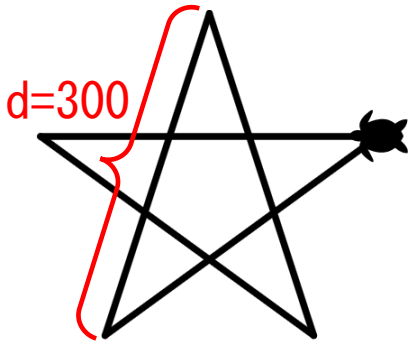
```
initializeTurtle()      # 描画エリアの初期化
bgcolor('yellow')       # 背景色を黄
color('green')          # 線色を緑
for i in range(4):      # 変数 i を 4 回繰り返す
    forward(200)        # 200 進む
    left(90)            # 左に 90 度回転
```

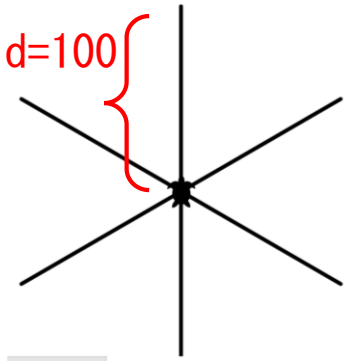


⑥課題:1 辺の長さdであるn角形を描画するコードを考えよ。その上で 1 辺の長さ 100 の正7角形を描画せよ。

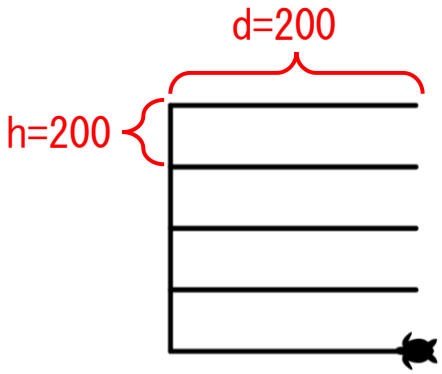


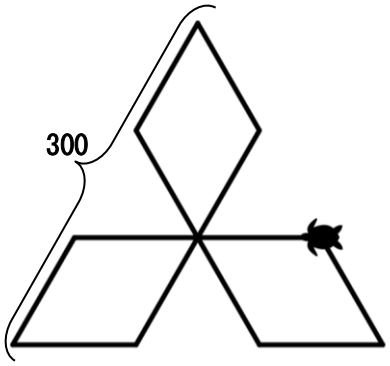
⑦ 課題:次の図形を描画するコードを考えよ。





課題:次の図形を描画するコードを考えよ。





https://www.mitsubishi.com/ja/profile/group/mark/より引用

図形描画解答例

```
left(90)
        for i in range(4):
          forward(200)
          left(90)
          forward(50)
          right(90)
          forward(-200)
```

```
        for i in range(3):
          forward(200)
          left(60)
          forward(100)
          left(120)
          forward(100)
          left(60)
```

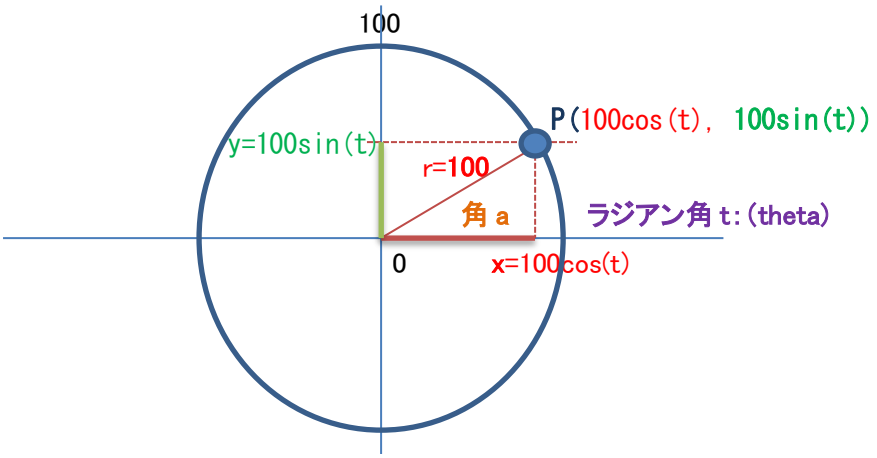
```
n=9
d=100
for i in range(n):
  forward(d)
  left(360/n)
```

⑨ 例題：でたらめな軌跡

```
import random                    # 乱数
for i in range(50):              # 以下を 50 回繰返し
  forward(random.randint(1,50))  # 1～50 の間ででたらめに前に進む
  left(random.randint(1,360))    # 1～360° の間ででたらめに左回り
```

⑩ 円　例：中心座標(300,250)、半径 r=100 の円…数学で円の関数、弧度法（ラジアン）に関して未履修の[illegible]ャンプしても結構です。

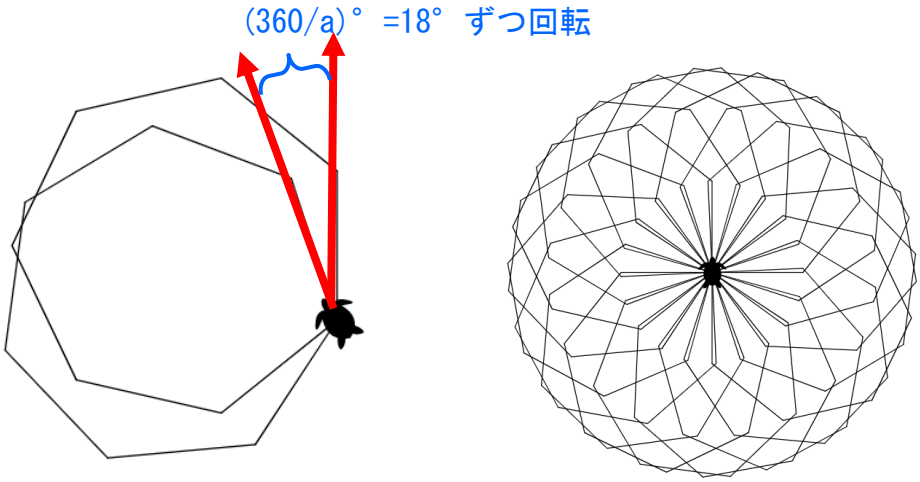
```
initializeTurtle()   # 先ず中心座標(0,0)、半径 100 の円周上の点 P の座標を(x,y)とする。
speed(10)            # 描画処理を速める（1～13）
for a in range(360):  # 角 a は、0~360 度の範囲で変化
  t = math.radians(a)  # 角 a のラジアン角を t とする
  x = 100*(math.cos(t))  # 円周上 P のx座標
  y = 100*(math.sin(t))  # 円周上 P のy座標
  goto(x+300,y+250)
```



⑪ 反復 入れ子 乱数などを活用する幾何学デザイン CG

例題 課題⑥で描けるn角形を、a 回、(360/a)° ずつ左回転させて反復させよ。

a=20, d=100, n=7 を代入して描画せよ。




```
initializeTurtle()      # 初期化
bgcolor('white')        # 背景色 白
color('black')          # ペンカラー色 黒
speed(10)               # 10 倍速
width(1)                # ペン幅 1
a=20                    # 多角形を回す回数 20 回で 360°
d=100             # 多角形の辺の長さを 100 とする
n=7                     # 多角形を 7 角形とする
for j in range(a):      # 多角形を左に(360/a)° ずつ a 回回せ
  for i in range(n):      # n=7角形を描け
    forward(d)          # d=100 進む
    left(360/n)         # (360/7)° 左に回せ
  left(360/a)           # (360/20)° 左に回せ
```

課題 本例題をコピーして実装したうえで、オリジナルデザインを提出すること

INDEX

例題 ⑪-a

```
initializeTurtle()
speed(10)
width(1)
for j in range(36):
  for i in range(3):
    forward(200)
    left(120)
  left(10)
```

例題 ⑪-b

```
initializeTurtle()
pencolor('orange')
width(1)
for j in range(36):
  for i in range(4):
    forward(200)
    left(90)
  left(10)
```

例題 ⑪-c

```
initializeTurtle()
width(1)
for i in range(4):
  forward(200)
  left(90)
goto(350,350)
for i in range(4):
  forward(200)
  left(90)
goto(150,350)
goto(200,250)
goto(200,50)
goto(150,150)
goto(350,150)
goto(400,50)
```

例題 ⑪-v ⑪例題のプログラムに下の朱書き部分を追加

```
initializeTurtle()
import random                    # 乱数モジュールを追加
col=['red','purple','blue','green','yellow','white']   # 色を指定
speed(10)
width(1)
a=20
d=100
n=7
for j in range(a):
  for i in range(n):
    forward(d)
    left(360/n)
  left(360/a)
  c=pencolor(random.choice(col))  # 変数 j の反復ごとに乱数で色を選択して、変数 c に代入
  pencolor(c)            # 色は変数 c
```

例題 ⑪-e

```
initializeTurtle()
import random
col=[(255,255,255),(255,255,0),(255,0,255),(0,255,255),(0,0,255),(255,0,0),(0,255,0),(255,96,255),(96,255,255),(255,255,96)]
for m in range(36):
  c=pencolor(random.choice(col))
  pencolor(c)
  width(1)
  left(10)
  for i in range(5):
    forward(100)
    left(72)
```

例題 ⑪-f

```
initializeTurtle()
import random
col=[(255,255,255),(255,255,0),(255,0,255),(0,255,255),(96,96,255),(255,96,96),(96,255,96),(255,96,255),(96,255,255),(255,255,96)]
for i in range(6):
  c=pencolor(random.choice(col))
  pencolor(c)
  width(1)
  forward(10)
  right(60)
  forward(20)
  right(60)
  forward(30)
  right(60)
  forward(40)
  right(60)
  forward(50)
  right(60)
```

例題 ⑪-g

```
initializeTurtle()
col=(100,255,255);
for m in range(1):
  pencolor((col))
  width(1)
  left(1)
  for i in range(72):
    forward(100+i*5)
    forward(-(100+i*5))
    left(5)
```

例題 ⑪-h

```
initializeTurtle()
for i in range(72):
  t = math.radians(i*5)
  x = 100*(math.cos(t))
  y = 100*(math.sin(t))
  col=(100,100+i*2,255);
  pencolor((col))
  width(1)
  forward(20+i*5)
  forward(-(20+i*5))
  goto(300+x,250-y)
  left(5)
```

例題 ⑪-i

```
initializeTurtle()
for i in range(360):
  t = math.radians(i)
  x = 100*(math.cos(t))
  y = 100*(math.sin(t))
  col=(70+i*2,70+i*2,255);
  pencolor(col)
  width(1)
  forward(20+i*3)
  forward(-(20+i*3))
  goto(250+1.5*x,250+3*y/2)
  pendown
  left(5)
```

(10-2) グラフ描画　外部ライブラリ matplotlib ☞(11)①を先に確認すること　応用は(15)へ

① データをプロットして図示する場合

```
import matplotlib.pyplot as plt    #  matplotlib をインポートする
y=[1,50,25,42,88]                  # y
x=[1,2,3,4,5]                      # x
plt.plot(x,y)                      # グラフ描画
plt.title("sample")                # グラフタイトル
plt.xlabel('x')                    # 横軸ラベル
plt.ylabel('y')                    # 縦軸ラベル
plt.show()                         # 表示する
```

② 関数を図示する場合

```
import matplotlib.pyplot as plt    #  matplotlib をインポートする
import numpy as np                 #  numpy をインポートする
x = np.arange(0, 10, 0.1)          #  numpy arange 関数で x の範囲と刻み値設定
y = -1*x**2+2*x+200                #  y=-x^2+2x+200
plt.plot(x, y, c='green')          #  緑、実線
y = 3*x**2-2*x                     #  y=3x^2-2x
plt.plot(x, y, c='red',linestyle='--',) #  赤、破線
plt.show()                         # 表示する
```

(11) 配列 複数の値を一括で扱うことができる番号付きの収納箱　②～④は応用

(11-1) 基本　配列名=[要素 1,要素 2,要素 3,　]

```
x=['a','b','c']        # 配列
print(x[1])            # 何だった？
```

配列 0 番目	配列 1 番目	配列 2 番目
a	b	c

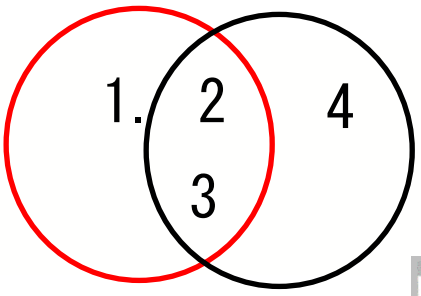
```
a=[1,4,7]              # 配列
print(a[0])            # 何だった？
```

	配列 1 番目	配列 2 番目
1	4	7

(12) 統計用 statistics モジュールへジャンプ

(11-2) 配列集合

```
a={1,2,3}          # 配列集合
b={2,3,4}          # 配列集合
print(a|b)         # 和集合
print(a&b)         # 積集合
print(a-b)         # 差集合
print(b-a)         # 差集合
```



(11-3) リスト list

```
list_x=[2,5,8]              # 整数型配列
print(list_x)

list_y=[1,1.5,3,'love']     # 整数型、小数型、文字型混合配列
print(list_y)               # 配列yを表示

list_y[0]=0.6               # 配列 y の「0 番にデータ 0.6 に変更」
print(list_y)               # 配列yを表示

list_y.insert(2,'yes')      # 配列 y の「2 番にデータ' yes' を挿入」
print(list_y)               # 配列yを表示
```

コードを**上から順番**に処理している　順次処理

List の並べ替えは(19)並べ替え ソート

(11-4) 辞書型 (データベース的な) 処理　キーkey👉値 value 呼び出し

key	value
210	川崎市川崎区幸区
211	川崎市幸区中原区
213	川崎市高津区
214	川崎市多摩区
215	川崎市麻生区
216	川崎市宮前区

(11-4a) サンプルデータの取得

http://strnun.fool.jp//pov-ray_strnun/kawasaki_postcode.CSV

から川崎市郵便番号表を取得し、サクラエディタ、atom、メモ帳などで開き、丸ごとコピーをする。

➡カンマ区切りのテキストデータ　csv　参照データベース RDB

(11-4b) 表作成、表に要素追加 修正

```
postcode={ }                # { } 内に取得した CSV を挿入
print(postcode[211])        # リストの場合は(k)「k 番目の値を返せ」だが、key[211]に紐づく値を返せ
postcode[230]='横浜市鶴見区'  # { } 内に要素を追加
print(postcode[230])        # key230 を表示
postcode[1]='日本 '         # { } 内に要素を追加
print(postcode)             # 表の要素全体を表示
postcode.pop(1)             # { } 内から要素を削除
print(postcode)             # 表の要素全体を表示
```

(11-5) ソート(並べ替え)

① アルゴリズムに基づくコーディング 大学共通テスト対策上必須だが非効率！

ア 昇順

```
ar = [3,8,6,1,9,5,4,2,7]          # 配列 [3,8,6,1,9,5,4,2,7]
n=len(ar)                         # データ個数
def bs(ar):                       # 関数 bs(bubble sort)
  for i in range(n-1):            # 変数 i を 0 から n-2 の範囲で繰り返す
    for j in range(0, n-i-1):     # 0 から n-2-i の範囲で繰り返す
      if ar[j] > ar[j+1]:         # 隣り合った要素を比較し左が大きければ
        ar[j], ar[j+1] = ar[j+1], ar[j]  # 順序入れ替え
bs(ar)                            # 配列 [3,8,6,1,9,5,4,2,7] をバブルソートする
print("newarray", ar)             # 'newarray' を表示
```

イ 降順

```
def bs_desc(ar):
  n = len(ar)
  for i in range(n-1):
    for j in range(0, n-i-1):
      if ar[j] < ar[j+1]:
        ar[j], ar[j+1] = ar[j+1], ar[j]
ar = [3,8,6,1,9,5,4,2,7]
bs_desc(ar)
print("newarray", ar)
```

② モジュールを活用したコーディング 大学共通テスト対策上不要だが高効率！

ア 昇順

```
list = [3,8,6,1,9,5,4,2,7]  # 配列 list
list.sort()          #  list を並べ替え(デフォルト:昇順)
print(list)          #  表示する
```

イ 降順

```
list.sort(reverse=True)     #  list を並べ替え(降順)
print(list)          # 表示する
```

課題 配列[9,-5,4.1,0,-2,4.2]を昇順降順でソートせよ。

```
list_p=[9,-5,4.1,0,-2,4.2]   # 配列 p
print(list_p)        # 表示する
list_p.sort()        # 昇順
print(list_p)        # 表示する
list_p.reverse()       # 降順
print(list_p)        # 表示する
```

(11-6) サーチ(探索)

① アルゴリズムに基づく順次探索コーディング1 大学共通テスト対策上必須だが非効率！

```
ar = [3,8,6,1,9,5,4,2,7]      # 配列 ar
x = 4                         # 変数 x
index = -1                    # 配列順番 index の初期値を-1 とする。
for i in range(len(ar)):      # 配列の個数回処理を反復
  if ar[i] == x:              # i 番目のデータを x として発見した場合
    index = i                 # index に i を代入し
    break                     # 反復を中止
if i == -1:                   # 発見できなかった場合は
  print("none")               # "none"を返す
else:                         # そうでなければ
  print("INDEX", i)           # "順番 INDEX"index の値を返す
```

② アルゴリズムに基づく順次探索2 大学共通テスト対策上必須だが非効率！

```
ar = [3,8,6,1,9,5,4,2,7]      # 配列 ar
x=1                           # 変数 x
for i in range(len(ar)):      # 配列の個数回処理を反復
  if ar[i] == x:              # i 番目のpデータを x として発見した場合
    print("INDEX", i)         #"順番 INDEX"index の値を返す
    break                     # 反復を中止
else:                         # 無ければ
  print('none')
```

③ アルゴリズムに基づく二分探索1 大学共通テスト対策上必須だが非効率！

```
ar = [3,8,6,1,9,5,4,2,7]      # 配列 ary を設定
x=7                           # 探索値を x とする
ar.sort()                     # 配列 ary を昇順ソート☜必須手順
l = 0                         # 配列左端すなわち最小値の index(場所)を変数 l = 0 とする
r = len(ar) - 1               # 配列右端すなわち最大値の index(場所)を変数 r = (データ個数-1) とする
while l <= r :                # 最小値～最大値の間で反復処理(条件 P)
  m = (l + r) // 2            # 中央値の index(場所)を変数 m とする
  if ar[m] == x :             # もし配列の中央値が探索値と同値ならば(条件 Q)
    print("INDEX",m)          # 結果を表示
    break                     # 反復処理を抜ける
  elif ar[m] < x:             # (条件 Q が false で)もし配列の中央値が探索値未満ならば(条件 R)
    l = m + 1                 # 最小値の index(場所)は中央値の index(場所)の右隣にする
  else:                       # (条件 Q･®が共に false)すなわち配列の中央値が探索値以上
    r = m -1                  # 最大値の index(場所)は中央値の index(場所)の左隣にする
print('none')                 # 反復処理(条件 P)を抜けた場合"none" を返せ
```

④ アルゴリズムに基づく二分探索２【関数使用】大学共通テスト対策上必須だが非効率！

```
def src(arr, x):                # 関数 src を設定
  arr = [3,8,6,1,9,5,4,2,7]     # 配列 arr を設定
  x=10                          # 探索値を x とする
  arr.sort()                    # 配列 arr を昇順ソート
  l = 0                         # 配列左端すなわち最小値の index（場所）を変数 l = 0 とする
  r = len(arr) - 1              # 配列右端すなわち最大値の index（場所）を変数 r =（データ個数-1）とする
  while l <= r:                 # 最小値～最大値の間で反復処理（条件 P）
    m = (l + r)//2              # 中央値の index（場所）を変数 m とする
    if arr[m] == x:             # もし配列の中央値が探索値と同値ならば（条件 Q）
      return m                  # 中央値の index（場所）を返せ
      break                     # 反復処理を抜ける
    elif arr[m] < x:            #（条件 Q が false で）もし配列の中央値が探索値未満ならば（条件 R）
      l = m + 1                 # 最小値の index（場所）は中央値の index（場所）の右隣にする
    else:                       #（条件 Q・®が共に false）すなわち配列の中央値が探索値以上
      r = m - 1                 # 最大値の index（場所）は中央値の index（場所）の左隣にする
  return ″none″                 # 反復処理（条件 P）を抜けた場合”none” を返せ
result = src(arr, x)            # 結果は関数から引っ張る
print('INDEX',result)           # 結果を表示
```

⑤ モジュールを活用したコーディング 大学共通テスト対策上不要だが高効率！

```
list = [3,8,6,1,9,5,4,2,7]
result = 0 in list          # 配列に 0 があるかの結果
print(result)               # 結果表示

list = [3,8,6,1,9,5,4,2,7]
result = 5 in list          # 配列に 5 があるかの結果
print(result)               # 結果表示

list = [4,5,2,1,3,5,3,5,1]
cnt = list.count(5)         # 配列にある 5 の個数
print(cnt)

index = list.index(3)       # 配列にある 3 の場所
print(index)

x=max(list)                 # 配列の最大値
print(x)

y=min(list)                 # 配列の最小値
print(y)
```

```
a=sum(list)              # 配列の合計値
print(a)

b=len(list)              # データの個数
print(b)

list = [3,8,6,1,9,5,4,2,7]
x=10
result = x in list
if result ==True:
  print(list.index(x))
else:
  print('none')
```

(11-7) 文字列サーチ(部分探索)➡for 文:シーケンスを反復処理

①リスト内包表記:配列から新しいリストを作成する方法

```
list = ['albert', 'carol', 'ester','jane','peter', 'george','william','Catherine','john','richard','Diana']
match = [a for a in list if "er" in a]
print(match)
```

② if 文を for ループ内で使用➡配列内の er を含む文字列を検索

```
list = ['albert', 'carol', 'ester','jane','peter', 'george','william','Catherine','john','richard','Diana']
new_list =[]
for x in list:
    if "er" in x:
        new_list.append(x)
print(new_list)
```

(11-8) 文字数

```
t='world wide web'      # 文字列定義
print(t)          # 文字表記
print(len(t))         # 文字数表記
```

(11-9) 二次元配列

```
list_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
list_2= [10, 20, 30, 40, 50, 60, 70, 80, 90]
list_3= [100, 200, 300, 400, 500, 600, 700, 800, 900]
list_123 = [
[1, 2, 3, 4, 5, 6, 7, 8, 9],
[10, 20, 30, 40, 50, 60, 70, 80, 90],
[100, 200, 300, 400, 500, 600, 700, 800, 900]
]
print(list_123, list_123[0][1])
```

(12) 統計用 statistics モジュール

(12-1) 基本

```
import statistics                  # 統計モジュール導入
a=[1,4,7]                          # 配列
print(len(a))                      # データ個数
print(sum(a))                      # データ合計
print(min(a))                      # データ最小
print(max(a))                      # データ最大
print(statistics.p variance(a))    # データ分散
print(statistics.pstdev(a))        # データ標準偏差
print(statistics.median(a)         # データ中央値
```

(12-2) 統計値算出とヒストグラム作成：表計算で実施した基礎統計を Python でプログラミング

配列 A より A.csv　　配列 B より B.csv　　二つを DL して配列 a=[ ]　配列 b=[ ]にインポート

```
import statistics                                          # 数理統計ライブラリ導入
import matplotlib.pyplot as plt                            # matplotlib.pyplot モジュールの読み込み
a=[42,25,36,38,55,14,65,67,78,63,39,59,57,86,53,75,48,45,86,29]   # 配列 a
b=[45,54,72,33,64,36,42,56,51,65,66,55,49,28,61,42,52,54,56,79]   # 配列 b
print(len(a))                                              # データ個数
print(sum(a))                                              # 合計
print(min(a))                                              # 最小
print(max(a))                                              # 最大
print(statistics.mean(a))                                  # 平均
print(statistics.p variance(a))                            # 分散
print(statistics.pstdev(a))                                # 標準偏差
print(statistics.median(a))                                # 中央値
print(statistics.mode(a))                                  # 最頻値

deviation_values = [ ]                                     # 偏差値の配列
for value in a:                                            # 変数 value 配列 a で反復
  deviation_value = int((value − statistics.mean(a)) / (statistics.pstdev(a)) * 10 + 50)
  deviation_values.append(deviation_value)
print(deviation_values)          # 偏差値の配列を出力する
```

偏差値=50+(値−平均値)*10/標準偏差

```
fig = plt.figure(figsize =(3,2))                           # グラフサイズ
plt.hist(a, bins = [0, 10, 20, 30,40, 50, 60, 70,80, 90, 100])   # 階級
plt.title("A Histogram")                                   # タイトル
plt.show()
```

配列 b についてもコーディングしてください

(12-3) 区間を以下のように取り、度数分布表から折れ線でヒストグラムを描画

該当度数分布表のデータを DL(csv)して配列に代入

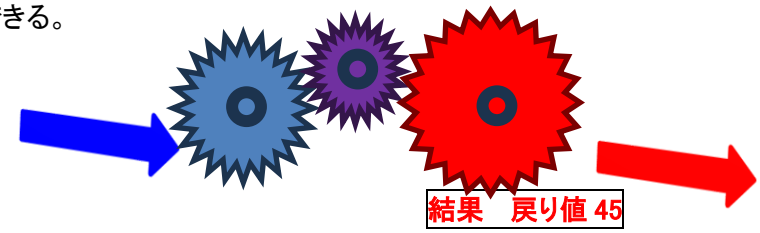
```
import matplotlib.pyplot as plt     # matplotlib.pyplot モジュールの読み込み
x=[0,10,20,30,40,50,60,70,80,90,100]  #  x 軸
a=[0,1,2,3,3,4,3,2,2,2,0,0]       # 配列 a
b=[0,0,1,2,4,7,4,2,0,0,0]         # 配列 b
plt.plot(x,a,c='purple')          #  a を可視化 c='purple' 省略　色は自動
plt.plot(x,b,c='red')             #  b を可視化 c='red'  省略　色は自動
plt.show()
plt.plot(a)に plt.plot(a,c=' purple ' )     # a を紫で可視化
plt.plot(a)に plt.plot(a,c=' red)           # a を赤で可視化
```

(13) 関数モジュール

関数：複数処理をまとめてワンアクションで実行　ここまでの範囲で Python が内蔵している int(x)、float(y)など実践している。加えて自分で作ることができる。

下例③の概念

1, 9 を渡す　引数



関数 1+2+3+4+5+6+7+8+9

(13-1) テキストを表示させる関数

```
def g ():          # 関数 g()を以下のように定義する
   print('good morning')  # good morning を表示せよ
g ()               # 関数 g を実行
```

(13-2) 税込み価格を計算する関数

```
def function(x):      # 関数 function(x)を定義
   price=x*1.1        # price は本体価格 x を 1.1 倍する税込価格
   return price       # 税込み価格（戻り値）を返せ
print(function(100))  # 本体価格を 100（引数）として関数処理を実行し戻り値を表示
```

(13-3) 合計値を求める関数を作成

解法1　参考：比較演算子の発展形 +=　➡ A+=B　A+B を A に返す意味　共通テスト向け

```
def add(x,y):              # 2 変数を x,y として関数 add を設定
   total=0                 # 合計値の初期値 0
   for i in range(x,y):    # 変数 i は x 以上 y 未満で反復
      total += i           # 合計値 total に変数 i を加えて合計値とする
   return total            # 合計値 total を戻り値とする
print(add(1,10))           # x=1、y=10 として add 関数を走らせて戻り値を表示
```

解法2　関数活用　➡ total return の字下げ位置の違いに注目

```
def add(x,y):                # 2 変数を x,y として関数 add を設定
   total=0                   # 合計値 total の初期値 0
   total=sum(range(x,y))     # 合計値 total は x 以上 y 未満で合計せよ
   return total              # 合計値 total を返せ
print(add(1,10))             # x=1、y=10 として add 関数を走らせて戻り値を表示
```

(13-4) 2 次方程式における解の公式を実装

```
def g(a, b, c):              #  各項の係数を a,b,c として関数 g を設定
   D = (b*b - 4*a*c)**0.5    #  判別式部を D とする　直接 x1,x2 の式に入れてもよい
   p = (-b + D)/(2*a)        #  解 1
   q = (-b - D)/(2*a)        #  解 2
   return p,q                #  解を返せ
print(g(2,1,-6))             #  係数設定
```

$f(x)=ax^2+bx+c$

$f(x)=0$

$x\{=-b\pm(b^2-4ac)^{0.5}\}\ /2a$

発展 実数解と複素数解を判別 前提➡複素数を扱うライブラリ cmath

```
import cmath                      # ライブラリ cmath
def g(a,b,c):
   D = (b**2- 4*a*c)              #  判別式 D=b^2-4*a*c
   if D>0:or D==0:                #  判別式 D>=0ならば
      p = (-b+D**0.5)/(2*a)       # 解 p=(-b+D^0.5)/2*a
      q = (-b-D**0.5)/(2*a)       # 解 q=(-b-D^0.5)/2*a
      return p,q                     # 戻り値 p,q
   else:                          #  判別式 D<0ならば
      r= -b/(2*a)                 #  実数部 r=-b/2a
      i= ((D**0.5)/(2*a)).imag    #  虚数部 i=√D・i
      p= complex(r, i)            #  解 p=実数部+虚数部
      q= complex(r, -i)           #  解 q=実数部-虚数部
      return p,q                     # 解 p,q を返せ
a=float(input("a"))               # 係数 a のプロンプト
b=float(input("b"))               # 係数bのプロンプト
c=float(input("c"))               # 係数cのプロンプト
print(g(a,b,c))                   #
```

発展 (26-6)参照 極めつけの数学用ライブラリ sympy➡圧倒的なステップ数(行数)短縮

```
import sympy                      # ライブラリ sympy
a=float(input("a"))
b=float(input("b"))
c=float(input("c"))
x=sympy.Symbol('x')                  # ライブラリ sympy で x を変数定義
s=sympy.solve(a*x**2 + b * x + c)  # 方程式の種類を選ばない！注①
print(s)
```

注①　4 次方程式 $90x^4-405x^3-225x^2+1890x=0$ で試してみよう

(14) 統計用ライブラリ numpy 活用による統計値算出

```
import numpy as np                  #  numpy を np とする
a = np.arange(12).reshape(3, 4)   # 配列 a(～12)を 3 行 4 列とする
print(a.shape)                      # 配列 a
print(a)                            # 配列 a を表示
print(np.sum(a))                    # 合計
print(np.mean(a))                   # 中央
print(np.var(a))                    # 分散
np.std((a))                         # 標準偏差
```

(15) データの可視化・グラフ描画　matplotlib

(15-1)データをプロットして図示する場合

```
import matplotlib.pyplot as plt #  matplotlib をインポートする
y=[8,50,25,42,88]          # y
x=[1,2,3,4,5]              # x
fig = plt.figure(figsize =(4,2))    # グラフサイズ
plt.plot(x,y,color= 'blue' )              # 折れ線グラフ描画
plt.bar(x,y,color= 'pink',width=0.2)    # 棒グラフ描画
plt.title("sample")                       # グラフタイトル
plt.xlabel( 'x' )                         # 横軸ラベル
plt.ylabel( 'y' )                         # 縦軸ラベル
plt.show()                                # 表示する
```

(15-2) 2データの可視化 折れ線グラフ

データ配列から折れ線で分布を比較描画➡データ配列 a,b をコピーして配列に代入

ヒストグラム: http://strnun.fool.jp/pov-ray_strnun/on_demand/sta1.mp4

```
import matplotlib.pyplot as plt       # matplotlib.pyplot モジュールの読み込み
x=[0,10,20,30,40,50,60,70,80,90,100]  # x 軸:区間
a=[0,1,2,3,3,4,3,2,2,0,0]             # 配列 a
b=[0,0,1,2,4,7,4,2,0,0,0]             # 配列 b
fig = plt.figure(figsize =(3,2))          # グラフサイズ(横、縦)
plt.plot(x,a,c= 'purple' )          # a を可視化 色:紫 c= 'purple'  省略:自動
plt.plot(x,b,c= 'red' )             # b を可視化 色:赤 c='red'
plt.show()
```

(15-3) 2データの可視化 マーカープロット

```
import matplotlib.pyplot as plt       # matplotlib.pyplot モジュールの読み込み
x=[0,10,20,30,40,50,60,70,80,90,100]
a=[0,1,2,3,3,4,3,2,2,0,0]
b=[0,0,1,2,4,7,4,2,0,0,0]
fig = plt.figure(figsize =(3,2))
plt.plot(x,a,'*', c='green' )  # 点プロット*
plt.plot(x,b,'o', c='brown')  # 点プロット●
plt.title("compare green:a,brown:b")
plt.show()
```

(15-4) 2データの可視化 折れ線グラフ+マーカープロット

```
import matplotlib.pyplot as plt       # matplotlib.pyplot モジュールの読み込み
x=[0,10,20,30,40,50,60,70,80,90,100]
a=[0,1,2,3,3,4,3,2,2,0,0]
b=[0,0,1,2,4,7,4,2,0,0,0]
fig = plt.figure(figsize =(3,2))
plt.plot(x,a,'*-', c='green')  # 点プロット(*) と折れ線
plt.plot(x,b,'o-', c='brown') # 点プロット(●)と折れ線
plt.title("compare green:a,brown:b")
plt.show()
```

(15-5) データの可視化 numpy 併用マーカープロット

```
import matplotlib.pyplot as plt       # matplotlib.pyplot モジュールの読み込み
import numpy as np              # numpy ライブラリの読み込み
x=[0,10,20,30,40,50,60,70,80,90,100]
a=[0,1,2,3,3,4,3,2,2,0,0]
b=[0,0,1,2,4,7,4,2,0,0,0]
fig, ax = plt.subplots()
ax.plot(x,a,'*',c='green')   # 点プロット*
ax.plot(x,b'o',c='brown')   # 点プロット●
plt.title("compare green:a,brown:b")
fig.set_figheight(2)   # グラフの高さ
fig.set_figwidth(3)   # グラフの幅
plt.show()
```

(15-6) データの可視化 別手法 numpy 併用

```
import matplotlib.pyplot as plt       # matplotlib.pyplot モジュールの読み込み
import numpy as np              # numpy ライブラリの読み込み
x=[0,10,20,30,40,50,60,70,80,90,100]
a=[0,1,2,3,3,4,3,2,2,0,0]
b=[0,0,1,2,4,7,4,2,0,0,0]
fig, ax = plt.subplots()
ax.plot(x,a, 'x-',c='green')   # ×と実線(-)
ax.plot(x,b, 's--',c='brown') # □と破線(- -)
plt.title("compare green:a,brown:b")
fig.set_figheight(2)            # グラフの高さ
fig.set_figwidth(3)             # グラフの幅
ax2.set_xlabel('number')       # 横軸ラベル
ax2.set_ylabel('value')        # 縦軸ラベル
ax.grid()                       # グリッド線
plt.show()
```

(15-7) データの可視化　numpy 併用グラフ分割

```
import matplotlib.pyplot as plt
import numpy as np
x=[0,10,20,30,40,50,60,70,80,90,100]
a=[0,1,2,3,3,4,3,2,2,0,0]
b=[0,0,1,2,4,7,4,2,0,0,0]
fig, (ax1, ax2) = plt.subplots(1, 2, sharey=True)　# y 軸共通で ax1, ax2 に分割
ax1.plot(x, a, 'o-',c='green')
ax2.plot(x, b, 'x--',c='brown' )
plt.title("compare green:a,brown:b")
fig.set_figheight(2)
fig.set_figwidth(8)
ax1.set_xlabel('number' )
ax1.set_ylabel('value')
ax1.grid()
ax2.grid()
plt.show()
```

(15-8) データの可視化　numpy 併用棒グラフ

```
import matplotlib.pyplot as plt
import numpy as np
x=np.array([0,10,20,30,40,50,60,70,80,90,100])　# numpy 配列
a=np.array([0,1,2,3,3,4,3,2,2,0,0])　# numpy 配列
b=np.array([0,0,1,2,4,7,4,2,0,0,0])　# numpy 配列
fig = plt.figure(figsize =(4,3))
plt.bar(x, a,width=5)　　# 棒グラフ(*) 、幅指定
plt.bar(x, b,width=3)　　# 棒グラフ(*) 、幅指定
plt.show()
```

(15-9) データの可視化　numpy 併用棒グラフ分轄

```
import matplotlib.pyplot as plt
import numpy as np
x=[0,10,20,30,40,50,60,70,80,90,100]
a=[0,1,2,3,3,4,3,2,2,0,0]
b=[0,0,1,2,4,7,4,2,0,0,0]
fig,(ax1,ax2) = plt.subplots(1,2,sharey=True)
ax1.bar(x,a,color='green',width=5)
ax1.set_title('a')
ax1.set_xlabel('number')
ax1.set_ylabel('value')
ax2.bar(x,b,color='red',width=5)
ax2.set_title('b')
```

```
fig.set_figheight(2)
fig.set_figwidth(5)
plt.show()
```

(15-10) データの可視化　円グラフ①

```
import matplotlib.pyplot as plt
x=[10,20,30,40,50,60,70,80,90,100]
label = ["A","B","C","D","E","F","G","H","I","J"]
fig = plt.figure(figsize =(3,3))
plt.pie(x,labels=label)
```

(15-11) データの可視化　円グラフ②

```
import matplotlib.pyplot as plt
x=[20,100,30,80,70,40,10,60,50,90]
label = ["A","B","C","D","E","F","G","H","I","J"]
fig = plt.figure(figsize =(3,3))
plt.pie(x,labels=label,autopct='%d%%') # 各要素名と割合%
plt.pie(x)
```

(15-12) データの可視化　円グラフ③

```
import matplotlib.pyplot as plt
x=[20,100,30,80,70,40,10,60,50,90]
label = ["A","B","C","D","E","F","G","H","I","J"]
fig = plt.figure(figsize =(3,3))
plt.pie(x,labels=label,autopct='%d%%') # 各要素名と割合%
plt.pie(x)
```

(15-13) データの可視化　円グラフ分轄④

```
import matplotlib.pyplot as plt
x=[20,100,30,80,70,40,10,60,50,90]
label = ["A","B","C","D","E","F","G","H","I","J"]
fig = plt.figure(figsize =(4,4))
plt.pie(x,startangle=90,counterclock=False,labels=label,autopct='%d%%')      //グラフ書き出し(A)を真上
plt.pie(x)
```

,counterclock=False　//False は右回り True は左回り

(15-14) データの可視化　円グラフ分轄⑤

```
import matplotlib.pyplot as plt
x=[20,100,30,80,70,40,10,60,50,90]
label = ["A","B","C","D","E","F","G","H","I","J"]
x,label = zip(*sorted(zip(x,label),reverse=True))       # 配列 x,ラベルを紐づけして降順並べ替え
fig = plt.figure(figsize =(4,4))
plt.pie(x,labels=label,startangle=90,counterclock=False,autopct='%d%%')
plt.show()
```

(15-15) ヒストグラム 表計算で実施した統計 分析のデータを用いた同等の実験 matplotlib 活用

度数分布表の可視化　区間を以下のように取り、度数分布表から折れ線でヒストグラムを描画

該当度数分布表のデータを DL(csv)して配列に代入 👉テキストエディタで開き配列をコピーする

```
import matplotlib.pyplot as plt     # matplotlib.pyplot モジュールの読み込み
x=[0,10,20,30,40,50,60,70,80,90,100]  #  x 軸
a=[0,1,2,3,3,4,3,2,2,0,0]       # 配列 a
b=[0,0,1,2,4,7,4,2,0,0,0]       # 配列 b
plt.plot(x,a,c='purple')        #  a を可視化 c='purple' 省略　色は自動
plt.plot(x,b,c='red')          #  b を可視化 c='red'  省略　色は自動
plt.show()
plt.plot(a)に plt.plot(a,c=' purple' )     # a を紫で可視化
plt.plot(a)に plt.plot(a,c=' red)        # a を赤で可視化
```

(15-16) 配列から直接ヒストグラムを作成　ax.hist

```
import matplotlib.pyplot as plt       #  matplotlib をインポートする
a = [42,25,36,38,55,14,65,67,78,63,39,59,57,86,53,75,48,45,86,29]  # 配列 a
b = [45,54,72,33,64,36,42,56,51,65,66,55,49,28,61,42,52,54,56,79]  # 配列 b
fig = plt.figure()              # 図を作成
ax = fig.add_subplot(1,1,1)         # グラフの配置指定(1,1,1)はデフォルト
ax.hist(a, bins=10, color='purple')     # ヒストグラム作成：得点配列と棒の数、色指定
ax.set_title('A')                  # グラフタイトル
ax.set_xlabel('Score')             # 横軸ラベル
ax.set_ylabel('Num')               # 縦軸ラベル
fig.show()                         # 表示する
```

課題　集団 B のケースで同様にヒストグラムを描画してください。

(16) ヒストグラムの応用 正規分布 ガウス曲線

(16-1) 乱数発生で得られたデータから正規分布を可視化 表計算学習時と同等の条件

```
import numpy as np                 #  numpy インポートし、np とする
import matplotlib.pyplot as plt    #  matplotlib をインポート
r = np.random.normal(              #  乱数
loc   = 5000,                      #  平均
scale = 100,                       #  標準偏差
size  = 10000,                     #  データの個数
)
fig = plt.figure()                 # データ可視化を変数 fig とする
ax = fig.add_subplot(1,1,1)        # デフォルト
ax.hist(r, bins = 50)              #  区間 50 でヒストグラム
ax.set_title('Gaussian curve')     #  グラフタイトル
ax.set_xlabel('Score')             # x軸 Score
ax.set_ylabel('Num')               # y軸 Num
fig.show()                         # グラフを表示
```

(16-2) 乱数による正規分布作成 2

```
import numpy as np          #  numpy インポートし、np とする
a = np.random.normal(       #  numpy で正規分布する乱数を発生させ a とする
loc   = 0,                  #  平均
scale = 1,                  #  標準偏差
size  = 1000000,            #  データの個数
)
Print(a)
```

(16-3) numpy、scipy を活用する正規分布ガウス曲線

```
import numpy as np              #  numpy インポート
import matplotlib.pyplot as plt      #  matplotlib をインポート
from scipy.stats import norm         #  NumPy の拡張版、数値解析ソフトウェア
import matplotlib.pyplot as plt      #  matplotlib をインポート
x = np.arange(-10,10,0.01)
y = norm.pdf(x,0,3)             # ガウス分布の確率密度関数,平均 0、標準偏差 3
plt.plot(x,y)
plt.xlim(-10,10)
```

(16-4) pandas(データ解析のためのライブラリ)活用 度数分布表

配列[47,36,39,42,37,43,46,44,33,42]を、階級の端値[30,35,40,45.45]で度数分布表

階級	階級値	度数
30~34.99	32.5	1
35~39.99	37.5	3
40~44.99	42.5	4
45~49.99	47.5	2

表計算ソフトによる解

```
import pandas as pd                                                    # pandas インポート
data = [47,36,39,42,37,43,46,44,33,42]                                 #  データ配列を定義
bins = [30, 35, 40, 45.45, 50]                                         # 階級の端値を定義
hist, bins = pd.cut(data, bins=bins, include_lowest=True, right=False, retbins=True) # ヒストグラム
freq_table = pd.value_counts(hist, sort=False).to_frame()
freq_table['階級の端値'] = bins[:-1] + (bins[1] - bins[0])/2             # 階級値
freq_table = freq_table.rename(columns={'index': '階級', 0: '度数'})     # 度数を計算
print(freq_table)                                                      # 表示
```

(17) 散布図 表計算で実施した統計 分析のデータを用いた同等の実験 ax.scatter

```
import matplotlib.pyplot as plt
x = [-3,-2,-1,0,1, 2, 3]          # 配列 x
y = [-7,-5,-3,-1,1,3,5]           # 配列 y
fig = plt.figure()                # figure を生成する
ax = fig.add_subplot(1, 1, 1)     # グラフの配置指定(1,1,1)はデフォルト
ax.scatter(x, y, c='red')         # 散布図を設定する c:color
plt.show()                        # 表示する
```

色分け

```
import matplotlib.pyplot as plt
plt.scatter(0,10,c='red')
plt.scatter(20,20,c='green')
plt.scatter(30,40,c='purple')
plt.show()
```

(18) 箱ひげ図 表計算で実施した統計 分析のデータを用いた同等の実験 ax.boxplot

(18-1) 1変量の場合

```
import matplotlib.pyplot as plt
a = [42,25,36,38,55,14,65,67,78,63,39,59,57,86,53,75,48,45,86,29] # 配列 a
fig = plt.figure()                # figure を生成する
ax = fig.add_subplot(1, 1, 1)     # グラフの配置指定(1,1,1)はデフォルト
ax.boxplot(a)                     #箱ひげ図を設定する
plt.show()                        # 表示する
```

(18-2) 2変量の場合

```
import matplotlib.pyplot as plt                                  # 描画ライブラリ
a = [42,25,36,38,55,14,65,67,78,63,39,59,57,86,53,75,48,45,86,29]  # 配列 a
b = [45,54,72,33,64,36,42,56,51,65,66,55,49,28,61,42,52,54,56,79]  # 配列 b
plt.boxplot([a,b], labels=["a","b"])                             # 箱ひげ図を設定する
plt.title("boxplot ")                                            # グラフ title
plt.show()
```

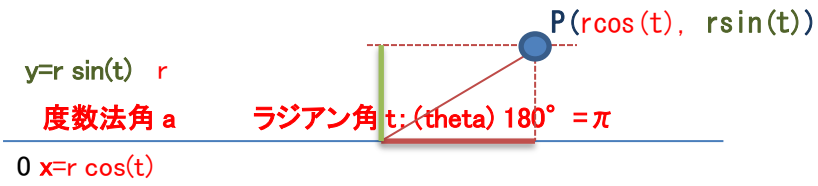
(19) math モジュール　数学…表計算ソフトウェアの関数的な処理

```
import math                      # math モジュールの導入
print(math.pi)                   # 円周率π
x=6.25                           # x＝6.25
print(math.ceil(x))              # 切り上げ
print(math.sqrt(x))              # 平方根
print(math.gcd(225,150))         # 最大公約数
y=10000                          # y=10000
print(math.log10(y))             # 対数　本例 log10 10000=4
kakudo=180                       # kakudo=180°（度数法）
print(math.radians(kakudo))      # kakudo を度数法➡弧度法変換　本例 180°＝π
print(math.sin(math.pi/6))       # 三角関数 sin
print(math.cos(math.pi/6))       # 三角関数 cos
print(math.tan(math.pi/6))       # 三角関数 tan
```

INDEX

参考

度数法-弧度法



(20) 移動平均　時系列データを平滑化する手法 Simple Moving Average➡トレンド分析

(20-1) 基礎実験 15 日間の変量推移である。2 日間移動平均と 3 日間移動平均を算出したテーブルを作成し、描画する。

date	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
value	10	6	2	4	9	11	7	3	2	7	12	5	1	3	8

```
date,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14
value,10,6,2,4,9,11,7,3,2,7,12,5,1,3,8
```

(20-2) 2 日間移動平均と 3 日間移動平均を算出

```
import pandas as pd              #  pandas を pd としてインポート
import matplotlib.pyplot as plt  #  matplotlib.pyplot を plt としてインポート
data={"date":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14],"value":[10,6,2,4,9,11,7,3,2,7,12,5,1,3,8]}
df = pd.DataFrame(data)
df["2-day movin'ave"]=df["value"].rolling(2).mean().round(1)
df["3-day movin'ave"]=df["value"].rolling(3).mean().round(1)
print(df)
```

INDEX

(20-3) 描画

```
plt.plot(df["date"], df["value"], label="daily")
plt.plot(df["date"], df["2-day movin'ave"], "k--", label="SMA(2)")
plt.plot(df["date"], df["3-day movin'ave"], "r--", label="SMA(3)")
plt.xticks(rotation=90)
plt.xlabel("date")
plt.ylabel("quantity")
plt.legend()
plt.show()
```

(20-4) 別方法：WEB から CSV データを取得する

```
import datetime                ☜datetime モジュール：日付や時刻
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
CSV_URL = "http://strnun.fool.jp/pov-ray_strnun/movin'ave2022data.csv"
df = pd.read_csv(CSV_URL, names=['date','value'], header=1);
df = df.dropna()
df['sma2'] = df.value.rolling(2).mean()
df['sma3'] = df.value.rolling(3).mean()
fig, axes = plt.subplots(2,1,figsize=(12, 10))
df[['value','sma2','sma3']].plot(ax=axes[0], xlim=[0, len(df)])
plt.show()
```

INDEX

(20-5) Python による東京都新規感染者数「短期（7 日）移動平均、長期（28 日）移動平均」分析

```
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline


CSV_URL = "http://strnun.fool.jp/pov-ray_strnun/covid19tkyvalue.csv" # データ;注2
df = pd.read_csv(CSV_URL, names=['date','value'], header=1);
df = df.dropna()
df['sma7'] = df.value.rolling(7).mean()
df['sma28'] = df.value.rolling(28).mean()
fig, axes = plt.subplots(2, 1, figsize=(12, 10))
df[['date', 'value','sma7','sma28']].plot(ax=axes[0], xlim=[0, len(df)])
plt.show()
```

注２；本件では筆者サーバ収納先にアクセスして取得

INDEX

(21) Python によるローソク足、チャートグラフ

参考 別項目「表計算を用いたチャート分析（ローソク足）」実行

(21-1) pandas の rolling 関数を用いる移動平均

```
import pandas as pd
import matplotlib.pyplot as plt
data = {"date":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14],"value":[10,6,2,4,9,11,7,3,2,7,12,5,1,3,8]}
df = pd.DataFrame(data)
df["2-day movin'ave"]=df["value"].rolling(2).mean().round(1)
df["3-day movin'ave"]=df["value"].rolling(3).mean().round(1)
print(df)
plt.plot(df["date"], df["value"], label="daily")
plt.plot(df["date"], df["2-day movin'ave"], "g", label="MA(2)")
plt.plot(df["date"], df["3-day movin'ave"],  "r", label="MA(3)")
plt.xticks(rotation=90)
plt.xlabel("date")
plt.ylabel("quantity")
plt.legend()
plt.grid(True)
plt.show()
```

(21-2) Web 上（当サイト）から取得したデータを可視化する。

```
pip install --upgrade pandas        # Web から値を取得する拡張ツールをインストール
pip install plotly                  # グラフ作成用ライブラリを pip コマンドでインストール
import plotly                       # plotly インポート
import plotly.graph_objs as gr      #  plotly.graph_objs を' gr' としてインポート
from plotly.offline import init_notebook_mode, iplot
import pandas as pd                 # pandas を' pd'  としてインポート
from datetime import datetime
df = pd.read_csv('http://strnun.fool.jp/pov-ray_strnun/covid19tky-220528.csv') # データ;注1
fig = gr.Figure(data=[gr.Candlestick(x=df['DATE'], # x 軸を列' 'DATE' としてローソク足描画
open=df['OP'],     # テーブルにおける「初値」のフィールド' OP'
high=df['MAX'],    # テーブルにおける「最大値」のフィールド' MAX'
low=df['MIN'],     # テーブルにおける「最小値」のフィールド' MIN'
close=df['CL'])])   # テーブルにおける「終値」のフィールド' CL'
fig.show()
```

注１；本件では筆者サーバ収納先にアクセスして取得

(21-3) 元データ取得

```
mport pandas as pd                  # pandas を' pd' としてインポート
from datetime import datetime
df = pd.read_csv('http://strnun.fool.jp/pov-ray_strnun/covid19tky-220528.csv') # データ;注1
df.tail()
```

(22) 多項式フィッティング

「我が国の人口推移回帰分析実験(総務省)」のデータを Python によって多項式にフィットさせる

(22-1)

A 配列からプロット

```
import numpy as np
import matplotlib.pyplot as plt
X =
np.array([  1920,1925,1930,1935,1940,1945,1950,1955,1960,1965,1970,1975,1980,1985,1990,1995,2000,2005,2010,2015,2020])
Y =
np.array([5596,5974,6445,6925,7193,7215,8411.4,9007.6,9430.1,9920.9,10466.5,11193.9,11706,12104.8,12361.1,12557,12692.5,12776.7,12805.7,12709.4,12577])
fig, ax = plt.subplots(dpi=100)
ax.scatter(X, Y, marker='o', color='red')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.grid()
```

B WEB 上から csv データ取得(本例は筆者サイト)してテーブル・配列表示

```
import pandas as pd
url = 'http://strnun.fool.jp/pov-ray_strnun/populationjapan.csv'      # csv データの所在を url とする
df = pd.read_csv(url)                                                # csv データを読む
print(df)                                                            # 表示
year = df['year'].values.tolist()                                    # year 列の値を変数 year とする
volume = df['volume(10k)'].values.tolist()                           # volume 列の値を変数 volume とする
print(year)                                                          # 配列 year 表示
print(volume)                                                        # 配列 volume 表示

import numpy as np                                                   # numpy 実装
import matplotlib.pyplot as plt                                      # matplotlib.pyplot 実装
X =np.array(year)                                                    # 配列 year を変数 X
Y =np.array(volume)                                                  # 配列 volume を変数 Y
fig, ax = plt.subplots(dpi=100)                                      # マーカープロット(サイズ 100)
ax.scatter(X, Y, marker='o', color='red')                            # 赤〇でプロット
ax.set_xlabel('$x$')                                                 # x軸
ax.set_ylabel('$y$')                                                 # y軸
ax.grid()                                                            # グリッド線
```

(22-2) 多項式の次数

```
DEGREE1 = np.polyfit(X, Y, 1)
DEGREE1
DEGREE2 = np.polyfit(X, Y, 2)
DEGREE2
DEGREE3 = np.polyfit(X, Y, 3)
DEGREE3
DEGREE4 = np.polyfit(X, Y, 4)
DEGREE4
```

(22-3) 1～4 次関数でフィッティングを実行

```
x = np.linspace(1920, 2020, 1000)
fig, ax = plt.subplots(dpi=100)
ax.scatter(X, Y, marker='o', color='red')
ax.plot(x, np.polyval(DEGREE1, x), 'black')
ax.plot(x, np.polyval(DEGREE2, x), 'brown')
ax.plot(x, np.polyval(DEGREE3, x), 'green')
ax.plot(x, np.polyval(DEGREE4, x), 'purple')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.grid()
```

(23) 相関/相関係数/散布図 covid-19 新規感染者数東京都-神奈川県（2019 年 4 月 1 日～2022 年 7 月 15 日）

(23-1) 配列としてデータ取り込み

```
import numpy as np
import seaborn as sns
sns.set()
```

tky =
np.array([67,98,92,118,141,85,87,156,183,199,198,174,100,159,127,151,206,186,109,101,123,123,134,170,119,82,41,113,47,59,165,154,93,87,57,37,23,39,36,22,15,27,10,30,9,14,5,10,5,5,11,3,2,14,8,10,11,15,21,14,5,13,34,12,28,20,26,14,13,12,18,22,25,24,47,48,27,16,41,35,39,34,29,31,55,48,54,57,60,58,54,67,107,124,131,111,102,106,75,224,243,206,206,118,143,165,286,293,290,188,168,237,238,366,260,295,239,131,266,250,367,462,472,292,258,309,263,360,461,429,331,197,188,222,206,389,385,260,161,207,186,339,258,256,212,95,182,236,250,226,247,148,100,170,141,211,136,181,116,76,170,149,276,187,226,146,80,191,163,171,220,218,162,97,88,59,193,195,269,144,78,211,194,234,196,205,107,65,176,140,248,203,248,146,78,166,177,284,183,235,132,78,139,145,185,186,201,124,102,158,171,220,203,215,116,87,209,122,269,242,294,189,156,293,316,392,374,352,255,180,298,486,531,522,539,390,315,188,402,482,570,561,419,312,370,501,532,459,584,328,301,354,574,601,596,621,481,305,457,681,822,664,736,556,397,570,758,895,890,954,708,491,869,961,1353,793,829,826,905,1315,1640,2520,2459,2332,1510,1252,1025,1480,1552,2044,1839,1595,1217,1253,1286,1485,1184,1079,986,619,1026,976,1065,871,770,634,393,556,676,734,577,639,429,276,412,491,434,307,369,371,266,350,378,445,353,327,272,178,275,213,340,270,337,329,121,232,316,279,301,293,237,116,290,340,335,304,330,239,175,300,409,323,303,342,256,187,337,420,394,376,430,313,234,364,414,475,437,446,354,248,397,554,543,537,570,421,308,511,591,728,666,758,541,404,710,842,860,758,875,633,425,830,927,1028,698,1052,877,708,605,618,590,908,1126,1033,574,924,971,1011,853,771,543,420,732,769,844,647,600,532,340,540,744,680,613,

541,446,259,471,487,509,472,435,350,237,370,440,436,435,467,302,208,337,500,452,453,388,375,236,436,620,570,563,535,386,317,476,714,674,660,715,518,342,592,920,896,823,951,613,503,829,1150,1309,1272,1411,1008,727,1391,1839,1984,1367,1148,1785,1449,2889,3234,3920,3386,4143,3134,2282,3798,4316,5149,4688,4652,4139,2972,2775,4425,5133,5908,5249,4358,3119,4534,5567,5729,5529,5247,4448,2537,4328,4334,4783,4350,3691,3124,1969,2926,3176,3097,2534,2370,1854,974,1634,1840,1677,1246,1276,1072,615,1011,1058,832,790,865,563,301,258,538,533,239,382,302,155,250,268,219,196,197,161,87,140,149,143,135,82,59,49,73,71,61,57,65,38,28,34,39,36,25,28,16,17,28,36,21,21,21,21,9,18,24,14,25,29,20,14,28,22,31,20,22,22,7,15,26,20,16,16,20,6,16,5,26,19,16,8,8,20,21,11,13,19,18,7,17,21,15,25,20,13,7,24,29,30,18,28,33,11,37,39,36,36,37,43,35,45,76,63,78,78,82,103,151,388,640,920,1223,1223,871,959,2196,3121,4055,4559,4166,3713,5181,7366,8634,9695,11217,9460,8493,12797,14077,16525,17610,17424,15893,11743,14432,21562,20663,19788,21110,17519,12194,17100,18275,18888,18648,11755,13061,10330,15515,17325,17856,16122,13509,12933,8801,11439,14559,10163,11123,11558,10314,9629,11802,12690,12246,10515,10804,9282,5370,8919,10812,10077,8456,9164,8127,4831,7831,10220,8456,7825,7441,6497,3853,3532,6426,8869,7284,7440,7843,4542,7846,9518,8220,7978,7390,7896,4382,6967,8650,8750,8110,8098,8024,4557,6920,8248,8535,6761,6794,5219,3477,5581,6771,6710,5393,5386,4935,3136,5042,6049,5392,3892,2978,3157,2397,3349,2997,2317,2678,3802,4709,3011,4447,4761,4214,4103,3798,3348,2374,3662,4353,4170,3571,3462,3313,2023,3268,3924,3388,2629,2549,2192,1342,2356,2415,2335,2111,2071,1584,1013,1800,1935,1876,1600,1526,1546,960,1528,2015,1819,1596,1681,1622,1076,1963,2329,2413,2181,2160,2004,1517,2514,3803,3621,3546,3616,3788,2772,5302,8341,8529,8777,9716,9482,6231,11511,16878,16662,19059])

kan =

np.array([19,25,31,21,27,6,18,66,26,55,76,31,15,20,40,56,33,44,30,17,13,25,39,32,31,16,7,11,25,26,15,36,23,10,14,7,13,7,8,13,7,15,10,32,16,12,5,8,8,21,10,7,5,5,1,3,5,8,10,4,6,1,3,2,6,6,5,5,0,1,0,1,3,6,0,3,2,7,6,2,5,0,0,1,4,6,7,4,10,6,31,8,11,24,20,21,11,8,23,25,32,35,23,16,28,42,47,43,49,30,11,30,68,53,28,18,33,14,33,70,76,53,58,72,48,89,81,118,107,128,81,38,27,66,123,117,136,79,51,84,95,104,82,101,49,39,56,85,66,75,106,64,50,59,76,81,108,67,79,29,63,106,112,82,68,53,15,52,101,65,78,68,60,20,38,11,58,78,90,65,17,58,79,78,66,59,37,11,65,54,65,77,63,37,29,54,88,79,85,50,49,23,47,88,55,90,60,60,22,64,64,71,60,65,65,22,68,44,109,104,137,77,36,98,130,147,146,147,114,61,133,226,205,208,192,162,70,67,159,252,219,214,149,83,158,214,197,188,191,134,65,152,245,213,285,223,231,121,226,287,319,295,314,238,188,348,346,494,466,479,343,334,394,432,587,470,380,365,412,622,591,679,838,995,727,694,905,767,984,871,829,794,957,737,716,731,627,521,553,351,394,386,433,385,397,390,221,187,234,224,288,201,164,121,141,176,178,154,105,108,71,133,115,142,129,131,100,96,97,93,119,116,162,131,52,84,138,138,131,113,119,59,100,124,124,107,95,109,55,91,93,159,111,107,77,56,72,128,121,117,102,64,93,96,136,133,133,129,142,68,100,118,175,168,180,132,94,114,205,242,209,247,220,142,157,252,318,225,216,221,159,212,257,255,242,275,247,222,214,151,224,229,303,338,237,277,319,337,339,328,296,199,248,269,308,327,268,266,218,200,225,227,260,258,233,139,159,218,215,234,224,249,173,179,201,189,220,247,170,141,160,210,184,231,181,162,135,163,201,192,221,231,203,192,181,209,211,230,254,226,180,198,250,322,355,310,389,280,308,361,403,446,539,460,412,433,521,630,652,547,531,539,758,1051,1164,1418,1580,1257,1686,1298,1484,1844,2082,1893,1860,2166,1572,1561,1807,2281,2356,2079,2584,2017,2021,2340,2878,2705,2524,2579,1946,2304,2632,2662,2377,2362,1719,1541,1921,1738,1868,1632,1242,971,738,1099,803,829,861,669,529,485,488,534,547,452,394,257,188,173,259,251,193,193,123,128,130,129,115,82,86,51,77,86,102,65,81,54,49,46,50,52,33,35,37,23,10,16,39,24,9,11,7,13,15,16,8,7,9,6,10,6,22,9,14,9,11,12,13,14,19,15,12,18,9,18,21,27,8,18,10,9,11,5,10,11,7,13,9,12,10,11,12,5,6,9,10,22,17,16,10,9,6,16,36,23,29,22,17,13,12,37,23,29,36,26,10,21,32,19,20,21,34,55,93,152,251,351,443,518,386,548,843,1155,1538,1751,1864,1990,2291,3348,3413,3407,3799,5278,4139,4801,5946,6473,8700,6143,7004,7411,7581,7078,9004,9101,8411,6571,8816,7251,8692,8125,8040,7938,5725,7201,7041,8024,8976,7700,6813,6305,6260,5996,6487,6723,5742,5697,6403,6102,6205,7190,6472,6198,5393,5622,4655,5748,6041,5617,4771,6104,4205,6572,4510,5264,5088,4270,4370,3856,2351,2830,3742,3772,4848,3553,3525,3300,3793,3887,3653,4806,4244,3053,3314,3956,3817,3706,3791,4098,3781,4117,4516,3737,3931,4048,3234,2609,2766,2733,2783,2834,3360,2566,2807,2450,2850,2213,2221,2575,1929,1590,1028,1780,1277,1181,1661,

1987,1662,2202,2340,1841,2012,2077,1905,1640,1887,1928,1673,1858,1816,1917,1378,1919,1707,1357,1574,1468,1346,903,1109,1131,824,1027,1090,854,746,761,823,699,787,777,717,599,652,836,652,933,895,841,624,815,918,808,987,1111,1062,752,1091,1263,1242,1519,1748,1815,1814,1939,3036,3130,3676,4009,4303,4230,4991,6193,6155,7603])

correlation = np.corrcoef(tky,kan)

sns.jointplot(tky, kan, kind="reg")

print(correlation[0,1])

(23-2) Web 上（当サイト）から取得したデータを可視化する。

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

CSV_URL="http://strnun.fool.jp/pov-ray_strnun//covid19tky-kan.csv"
df = pd.read_csv(CSV_URL, names=['date','tky','kan'], header=0);
df.head(836)
plt.scatter(df['tky'], df['kan'])
```

(24) データベース的な処理

(24-1) ウェブ上から csv データの取得（本例：筆者サーバ　文科省新体力テスト結果例

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sb
CSV_URL="http://strnun.fool.jp/pov-ray_strnun//spt.csv"
df = pd.read_csv(CSV_URL, names=['grade','sex','height','weight','sitting height','Grip strength','sit up','forward bend','side jump','shuttle run','50m run','long jump','ballthrow'], header=0);
df.head(1000)
```

(24-2) 条件選択

(25-1)全文に以下を加える

```
df=df[(df['grade']=='2nd') & (df['sex']=='f')] # 学年列で2年かつ性別列で女子を選択
print(df.head(1000))
```

(24-3) 射影

(25-1)全文に以下を加える

```
df=df[(df['grade']=='2nd') & (df['sex']=='f')] # 同上選択条件
df_data=df.iloc[:, 0:8]              # フィールド 0～8 で射影
print(df_data.head(1000))
```

(24-4) シーボーンを用いるグラフ化　男女人数棒グラフ

(25-1)全文に以下を加える

```
sb.countplot(x='grade',data=df,hue='sex')
plt.title('m-f')
plt.ylabel('heads')
plt.show()
```

(24-5) シーボーンを用いるグラフ化　男女別各学年 50m 走箱ひげ図

(25-1)全文に以下を加える

```
sb.boxplot(x = df['grade'] + df['sex'],
y=df['50m run'],data=df,hue='sex')
plt.show()-
```

(24-6) シーボーンを用いるグラフ化　男子の 50m 走とボール投げの相関散布図


```
df_=df[df['sex']=='m']
plt.scatter(df['50m run'], df['ballthrow'])
plt.show()
```

(25)　2022 年 9 月時点での日米の長期国債利率、ドル円レートで固定し、同レートでで元金$10,000 を最長 10 年、複利の貯蓄をする場合のリターンをシミュレーションし、円ベースで比較する。

(25-1) 各条件に変数を与える

元金$10,000=¥1,440,000

米国利率 ir=0.037　3.7%　ir(interest rate　利率)

日本利率 ir=0.0025　0.25%　jr(japan)

元金 x=10000, y=1440000

米国利息 i　(interest)

日本利息 j

反復回数 n

(25-2) 実装

```
x=10000
ir=0.037
for n in range(10):
  i=x*ir
  x=x+i
  print('you will get $',int(x),n+1,'years after')

y=1440000
jr=0.0025
for n in range(10):
  j=y*jr
  y=y+j
  print('you will get yen',int(y),n+1,'years after')elif (y==0 and x==2) or (y==1 and x==0)
```

(26) ライブラリ sympy を用いる高校数学の代数計算 数式処理

(26-1) 関数定義と値の代入

```
import sympy                    #  sympy を定義
x = sympy.Symbol('x')           #  変数 x を定義
y = sympy.Symbol('y')           #  変数 y を定義
f = x**2 + y**2 -9              #  関数 f=x^2+y^2-9
print(f.subs([(x, 1), (y, 2)]))       #  x=1、y=2 を代入
```

(26-2) 関数定義と展開

```
import sympy
x = sympy.Symbol('x')          #  変数 x を定義
y = (x + 2)*(x - 3)            #  元式を定義
y_ex = sympy.expand(y)         # 展開
print(y_ex)
```

または

```
from sympy import *
x = Symbol('x')
y=(x + 2)*(x - 3)
print(diff(y,x))
print(expand(y))
```

(26-3) 関数定義と因数分解

```
import sympy
x = sympy.Symbol('x')          # 変数 x を定義
y_ex = x**2 - x - 6            # 元式を定義
y_factor = sympy.factor(yf_ex)    # 因数分解
print(y_factor)
```

または

```
from sympy import *
x = Symbol('x')
y_ex = x**2 - x - 6
print(factor(y_ex))
```

(26-4) 関数定義と微分

```
import sympy
x = sympy.Symbol('x')          # 変数 x を定義
y = x**2 - x - 6               # 元式を定義  y=x²-x-6
print(sympy.diff(f))           # 微分
```

同義

```
from sympy import *
x = Symbol('x')
y = x**2 - 3*x - 6
print(diff(y,x))
```

三角関数のケース

```
import sympy
x = sympy.Symbol('x')
y = sympy.sin(x)
print(sympy.diff(y,x))
```

同義

```
from sympy import *
x = Symbol('x')
y = sin(x)
print(diff(y,x))
```

対数のケース 3 行目元式を差し替える

```
y = log(x)
```

課題 元式 $y=(2x-5)^6$

① 微分式 y‘を単項式で答えよ

② ①を展開せよ

```
from sympy import *
x = Symbol('x')
y=(2*x-5)**6
print(diff(y,x))
print(expand(y))
```

(26-5) 関数定義と積分

元式 $y=\int_3^4(x^2-x-6)\,dx$

```
import sympy
x = sympy.Symbol('x')          # 変数 x を定義
y = x**2 - x - 6               # 元式を定義 y=x^2-x-6
print(sympy.integrate(y))      # 不定積分
```

```
import sympy
x = sympy.Symbol('x')
y = x**2 - x - 6
print(sympy.integrate(y,(x,3,4)))          # 定積分
```

同義

```
from sympy import *
x = Symbol('x')
y = x**2 - x - 6
print(integrate(y))
```

同義

```
from sympy import *
x = Symbol('x')
y = x**2 - x - 6
print(integrate(y,(x,3,4)))
```

課題 元式 $y=\int_0^{\pi/12}\sin^2x\cos^2x\,dx$

① 定積分を計算せよ。 ②不定積分の式を答えよ。

(26-6) 二次方程式解法　参考: (13-4) 二次方程式解の公式実装

例題 $X^2-3x+2=0$ を解く

```
import sympy
x = sympy.Symbol('x')          # 変数、式を定義
ans=sympy.solve(x**2 - 3 * x + 2)  # 方程式を解く
print(ans)
```

(26-7) 連立方程式解法

$$\begin{cases} 3x+5y=29 \\ X+y=7 \end{cases}$$

```
import sympy          #  代数計算 数式処理ライブラリ
x = sympy.Symbol('x')     #  変数 x を定義
y = sympy.Symbol('y')     #  変数 y を定義
f = 3 * x + 5 * y - 29    #  関数 f=3x+5y-29
g = x + y - 7          #関数 f=x+y-7
print(sympy.solve([f, g]))    #  f=0、g=0 の方程式を解く
```

(26-8) 二つの関数の交点座標を求め、グラフで示す。

$$\begin{cases} y=x^2 \\ y=2x+24 \end{cases}$$

```
import matplotlib.pyplot as plt     #グラフ描画ライブラリ
import numpy as np                  # 統計用ライブラリ
import sympy                        # 代数計算 数式処理ライブラリ
x = sympy.Symbol('x')
y = sympy.Symbol('y')
f = x ** 2 - y                      # f=x^2-y
g = 2 * x +24 - y                   # g=2x+24-y
print(sympy.solve([f, g]))          # f=0、g=0 の方程式を解く
x = np.arange(-10, 10, 0.1)         # x 座標(-10～+10)
plt.plot(x, x ** 2, c='blue')       # y=x^2 式のグラフ描画
plt.plot(x, 2 * x + 24 , c='red')   # y=2x+24 式のグラフ描画
plt.show()
```

課題

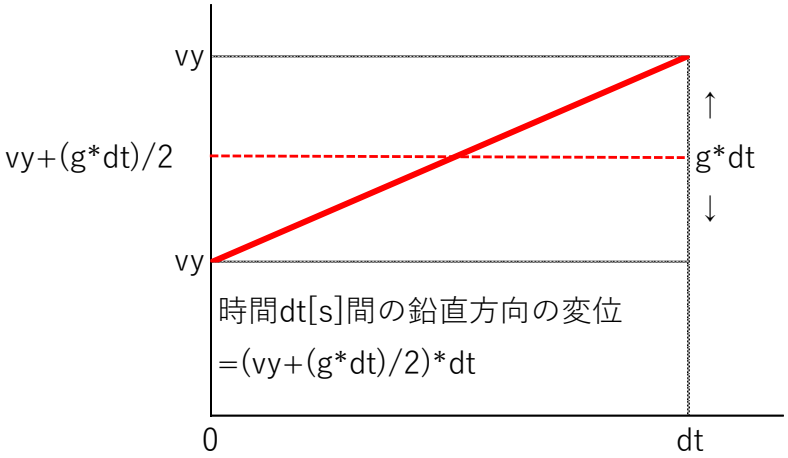
$$\begin{cases} y=x^2+4x-6 \\ y=-x^2+8x+10 \end{cases}$$

```
import sympy
import matplotlib.pyplot as plt
import numpy as np
x = sympy.Symbol('x')
y = sympy.Symbol('y')
f = x ** 2 + 4 * x -6 - y
g = -1 * x ** 2 + 8 * x + 10 - y
print(sympy.solve([f, g]))
x = np.arange(-10, 10, 0.1)
plt.plot(x, x ** 2 + 4 * x -6 , c='green')
plt.plot(x, -1 * x ** 2 + 8 * x + 10, c='red',linestyle='--',)
plt.show()
```

(27)（物理力学応用）下記の条件で物体を斜め上方に投げ上げる時の、放物運動をシミュレートして図示体験の意味で取り組んでください。【物理履修者が望ましい】

初速 vo=40[m/s] ,投げ上げ角度 a=45° ,重力加速度 g=9.8[m/s^2] ,初期位置 x=0[m],y=0[m]

アルゴリズム(問題解決手法)のヒント



時間上の微少区間を dt[秒]とすると、その間の鉛直方向速度 vy の変化分は g*dt
プログラミング上の記述は
vy= vy+ g*dt
また、dt[秒]の鉛直方向変位は
(vy+(g*dt)/2)*dt・・・左図赤破線下の長方形の面積
よって、鉛直方向変位は
y=y+(vy+(g*dt)/2)*dt

(27-1) 変位

```
import math                    # 数学モジュール
import matplotlib.pyplot as plt  # グラフ描画ライブラリ
dt=0.1                         # 時間微少区間を dt=0.1[s]とする　任意
v0=40                          # 初速を vo=40[m/s]とする　任意
a=45                           # 投げ上げ角度 a=45°とする　任意
g=9.8                          # 重力加速度 g=9.8[m/s²]とする
x=0                            # 初期位置 x=0[m]とする　任意
y=0                            # 初期位置 y=0[m]とする　任意
A=a*math.pi/180                # 投げ上げ角度 a を弧度法変換し、変数 A とする
vx=v0*math.cos(A)              # 水平方向の速度 vx=v0cos(A)
vy=v0*math.sin(a)              # 鉛直方向の速度 vx=v0sin(A)
for n in range(100)            # 100 回反復
  plt.scatter(x,y,color='red')     # 散布図を赤で
  vy=vy-g*dt                   # 微少時間 dt[s]後の鉛直方向の速度 vy
  x=x+vx*dt                    # 水平方向の変位 x
  y=y+(vy+(g*dt)/2)*dt         # 鉛直方向の変位 x　上記ヒント
  if y<0:                      #y が 0 未満の場合は、実行させない
    break                      # 反復停止
plt.show()
```

(27-2) 鉛直方向速度

```
dt=0.1                         # 時間微少区間を dt=0.1[s]とする　任意
v0=40                          #初速を vo=40[m/s]とする　任意
a=45                           # 投げ上げ角度 a=45°とする　任意
g=9.8                          # 重力加速度 g=9.8[m/s²]とする
x=0                            # 初期位置 x=0[m]とする　任意
y=0                            # 初期位置 y=0[m]とする　任意
A=a*math.pi/180                # 投げ上げ角度 a を弧度法変換し、変数 A とする
vx=v0*math.cos(A)              # 水平方向の速度 vx=v0cos(A)
vy=v0*math.sin(A)              # 鉛直方向の速度 vx=v0sin(A)
```

```
for n in range(50):                # 50 回反復
  plt.scatter(x,vy,color='blue')   # 散布図を青で
  vy=vy-g*dt
  x=x+vx*dt
plt.show()
```

(28) 三角関数の定理

(28-1) 正弦の加法定理

sin(A+B)=sinAcosB+cosAsinB　case-1

sin(A-B)=sinAcosB-cosAsinB　case-2

case-1　sin75°

```
import math                        # 数学モジュール
x = 75                             # ターゲットを 75° とする
a = 45                             # 角度 a を 45° とする
b = 30                             # 角度 b を 30° とする
r = math.radians(x)                 # x を弧度法変換して r とする
A = math.radians(a)                 #  a を弧度法変換して A とする
B = math.radians(b)                # b を弧度法変換して B とする
ans = math.sin(r)                   # sin75° の値を ans とする
print('sin(x) = ', round(sns, 2))    # sin75° の値を小数点 2 位まで表示
ANS = math.sin(A) * math.cos(B) + math.cos(A) * math.sin(B)       # sinAcosB+cosAsinB の値 ANS
print('sin',x,'=',round(ANS, 2))      # sinAcosB+cosAsinB の値の値を小数点 2 位まで表示
```

case-2　sin15°

```
import math                                # 数学モジュール
x = 15                                     # ターゲットを 75° とする
a = 45                                     #  角度 a を 45° とする
b = 30                                     # 角度 b を 30° とする
r = math.radians(x)                        #  x を弧度法変換して r とする
A = math.radians(a)                        #  a を弧度法変換して A とする
B = math.radians(b)                        #  b を弧度法変換して B とする
ans = math.sin(r)                          #  sin75° の値を ans とする
print('sin(x) = ', round(sns, 2))          #  sin75° の値を小数点 2 位まで表示
ANS = math.sin(A) * math.cos(B) - math.cos(A) * math.sin(B)          #  sinAcosB-cosAsinB の値 ANS
print('sin',x,'=',round(ANS, 2))           #  sinAcosB-cosAsinB の値を小数点 2 位まで表示
```

(28-2) 余弦の加法定理

cos(A+B)=cosAcosB−sinAsinB　case-1

cos(A−B)=cosAcosB+sinAsinB　case-2

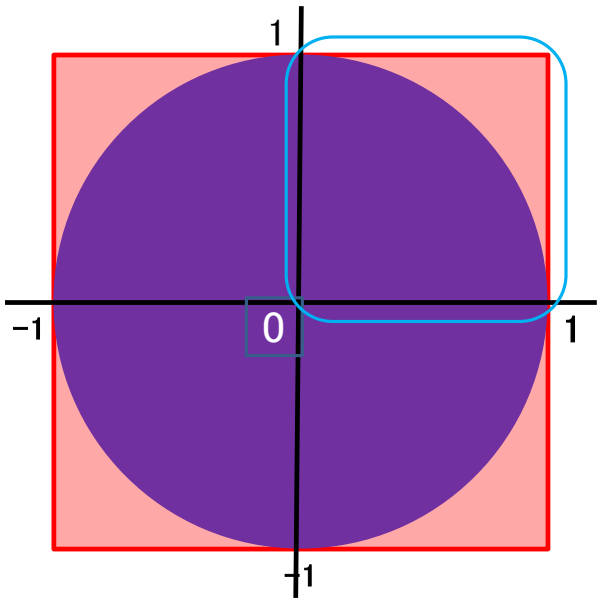
case-1　cos75°

```
import math
x = 75
a = 45
b = 30
r = math.radians(x)
A = math.radians(a)
B = math.radians(b)
ans = math.cos(r)
print('cos(x) = ', round(ans, 2))
ans = math.cos(A) * math.cos(B) - math.sin(A) * math.sin(B)
print('cos',x,'=',round(ans, 2))
```

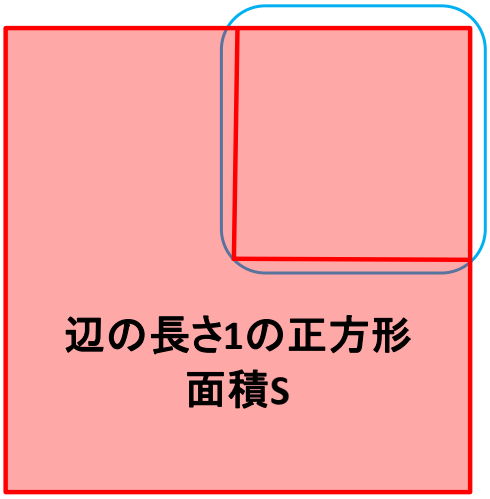
case-2　cos15°

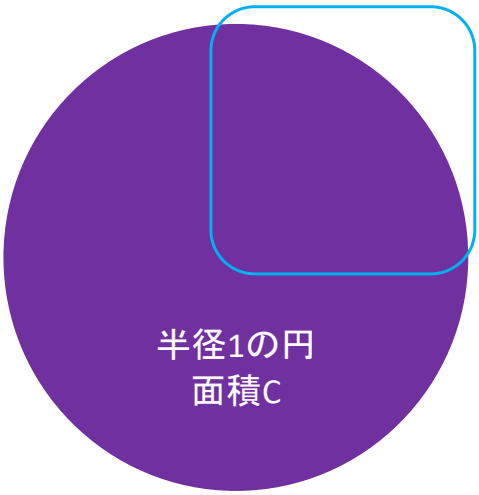
```
import math
x = 15
a = 45
b = 30
r = math.radians(x)
A = math.radians(a)
B = math.radians(b)
ans = math.cos(r)
print('cos(x) = ', round(ans, 2))
ans = math.cos(A) * math.cos(B) + math.sin(A) * math.sin(B)
print('cos',x,'=',round(ans, 2))
```

(29) 確率的シミュレーション：モンテカルロ法・・・参考資料は東北大学理学部様 web ページリンク引用



左図の第 1 象限(青線部)に着目する
横軸 x、縦軸 y
0 < = x < 1　　0 < = y < 1
1 辺の長さ 1 の正方形(赤)の面積 S=1*1=1
半径 1 の円(紫)の面積 C=π/4
全打点数 N
円内の打点数 M
M ：N = C ：S = π/4 ：1
π = 4M / N





モンテカルロ法①

```
import random                  # 乱数モジュール
import math                    # 数学用モジュール➡省略時は★を r = (x**2 + y**2)**0.5 にする
N = 1000000                    # 点の総数
M = 0                          # 円内に入った点の数
for i in range(N):             # x,y 座標を 0～1 までの乱数で生成、ランダムに打点
    x = random.random()
    y = random.random()
    r = math.sqrt(x**2 + y**2) # 点と原点との距離を計算　★
    if r <= 1:                 # 距離が 1 以下なら円内に入っていると判定
        M += 1
pi = 4 * M / N                 # 円周率の近似値を計算
print("円周率の近似値:", pi)    # 結果を表示
```

INDEX

モンテカルロ法②

```
import random
def estimate_pi(n):
    num_points_circle = 0
    num_points_total = 0
    for _ in range(n):
        x = random.uniform(0, 1)
        y = random.uniform(0, 1)
        distance = x**2 + y**2
        if distance <= 1:
            num_points_circle += 1
        num_points_total += 1
    return 4 * num_points_circle / num_points_total
print(estimate_pi(100000))
```

モンテカルロ法③

```
import random
import math
import matplotlib.pyplot as plt
N = 1000000                    # 点の総数
M = 0                          # 円内に入った点の数
x_list = []                    # x,y 座標のリスト
y_list = []
for i in range(N               # ランダムに点を打つ
    x = random.random()        # x,y 座標を 0〜1 までの乱数で生成
    y = random.random()
    r = math.sqrt(x**2 + y**2) # 点と原点との距離を計算
    if r <= 1:                 # 距離が 1 以下なら
        M += 1                 # 円内に入っていると判定して M に 1 を加えて代入
    x_list.append(x)           # x,y 座標をリストに追加
    y_list.append(y)
pi = 4 * M / N                 # 円周率の近似値を計算
print("円周率の近似値:", pi) # 結果を表示
```

モンテカルロ法④ プロット描画

```
def plot_points(n):
    x_inside, y_inside, x_outside, y_outside = [], [], [], []
    for _ in range(n):
        x = random.uniform(0, 1)
        y = random.uniform(0, 1)
        distance = x**2 + y**2
        if distance <= 1:
```

```
            x_inside.append(x)
            y_inside.append(y)
        else:
            x_outside.append(x)
            y_outside.append(y)
    fig, ax = plt.subplots()
    ax.scatter(x_inside, y_inside, color='red')
    ax.scatter(x_outside, y_outside, color='blue')
    plt.show()
plot_points(10000)
```

(30) 複素数 import cmath　　複素数を扱うライブラリ

```
import cmath
x=1+2j          # x=1+2j とする。(虚数部は j とすること)
y=3+4j          # y=3+4j とする。
print(x.real,x.imag)  # x の実数部と虚数部を表示せよ
print(x+y,y-x,x*y)  # x+y,y-x,x*y を計算せよ
```

(31) モデル化シミュレーション「待ち時間」

```
import random                          # 乱数モジュール
arrive=0                               # 到着時刻 arrive
start=0                                # レジ会計・治療など作業開始時刻 start
end=0                                  # レジ会計・治療など作業終了時刻 end
waitingtime=[ ]                        # 到着から開始までの待ち時間配列を設定 waiting time
for i in range(20):                    # 20 人分反復
  arrive=arrive+random.randint(1,5)  # 到着時刻は先着者の後 1～5 分後で乱数
  if end<=arrive:                      # 到着時刻が先客の終了時刻より後ならば
    start= arrive                      # 開始時刻は到着時刻
  else:                                # そうでないなら
    start=end                          # 開始時刻は先客の終了時刻
  service=random.randint(5,10)         # レジ会計・治療など作業経過時間は乱数で 5～10 分➡service
  end=start+service                    # 終了時刻は開始時刻に作業時間を加える
  waitingtime.append(start-arrive)     # 待ち時間配列に(開始時刻-到着時刻)を追加
  print(i+1, arrive,waitingtime[i],start,service,end) #順番、到着時刻、待ち時間、開始時刻、サービス時間、終了時刻表示
  i=i+1                                # 次番を処理
print('平均待ち時間',sum(waitingtime)/len(waitingtime))   # 待ち時間の平均値を表示
```

(32) コイントスシミュレーション

① x回の試行で表(裏)の出た回数を積算して試行回数で割る

```
import random
face = 0
x=2000
for i in range(x):
  if random.randint(0, 1) == 1:
    face += 1
print("omote: ", face)
print("ura: ", x-face)
print("omote rate: ", face / x)
```

② 結果を配列に取り込んで処理

```
import random
A=[]
x=600
def coin_toss():
  return random.randint(0, 1)
results = [coin_toss() for i in range(x)]
heads = results.count(1)
probability = heads / len(results)
print(probability)
```

③ ヒストグラムで可視化

```
import matplotlib.pyplot as plt          # 可視化モジュール
results = []                             # コイントス結果の配列を設定
for i in range(100):                     # コイントス試行回数を設定
    results.append(random.randint(0, 1)) # 乱数で 0,1 を発生
    head = results.count(1)              # 1 の出た回数を heads(表)
    probability = head / len(results)    # 表の回数/全試行回数を probability
print(probability)                       # probability を表示
plt.hist(results, bins=2)                # 結果をヒストグラム描画
plt.show()
```

④ 以下の仕様をコーディングする

・コイントスを n 回行う
・表 face が出現する回数を face とする
・確率を p とすると p=face/n
・この試行を m 回実行して、その都度確率 p を配列 result に入れる
・m 回試行分の配列からヒストグラムを作成
・その平均値 average を求めて表示

```
import matplotlib.pyplot as plt
result = []                          # ヒストグラムを描画するためのリスト
m=100                                # 試行回数
n=1000                               # 打点数
for i in range(m):                   # シミュレーションを m 回繰り返す
    face = 0                         # 表が出る回数 face 初期値 0
    for i in range(n):               # コイントスを n 回反復試行
        if random.randint(0, 1) == 1:  # 整数 0,1 の乱数発生で 1 が出現したら
            face += 1                # face に 1 を加えて代入
    p = face / n                     # 表が出る確率を算出を p とする
    result.append(p)                 # p を配列 result に追加
plt.hist(result, bins=20)            # ヒストグラムを描画
plt.show()
average = sum(result) / m            # 配列の合計値を試行回数で割って平均を求める
print("平均値は", average)           # 平均値を表示
```