1　基本操作

評価	C	B	A
基準	B に満たない.	各プログラムについて理解し, 例文を正しく実行することができる.	練習問題を, 例題を参考に正しく実行することができる.

1.1　表示・入力・代入

print(　) で, () の中身を表示することができる. また, 文字列の場合は, ' ' でくくる.
また, 人が入力した文字を読み取る場合は, input() を使う.
プログラミングにおいて, 変数を設定し数を代入して, 演算処理を行うことができる. = を用いることで, 数値代入をすることができる.

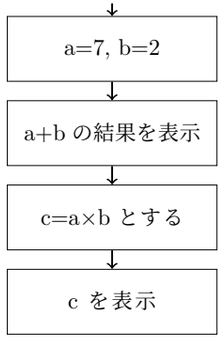
```
1 print('Hello World')
2 print(321)
```

1.2　演算

演算子は以下の通り. 例は, $a = 5, b = 2$ とする.

	演算子	記入例	結果
足し算	+	$a+b$	7
引き算	−	$a-b$	3
掛け算	*	$a*b$	10
割り算	/	a/b	2.5
商	//	$a//b$	2
余り	%	$a\%b$	1
累乗	**	$a**b$	25

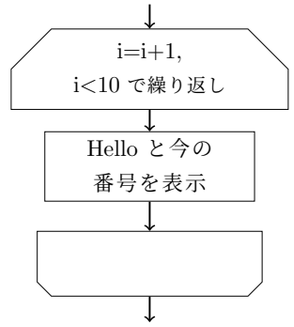
```
1 a=7;b=2
2 print(a+b)
3 c=a*b
4 print(c)
```



1.3　繰り返し

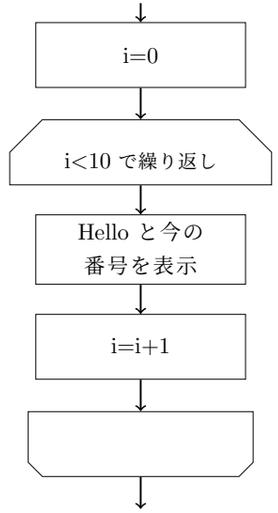
同じ操作を繰り返す際に用いる. for 文, while 文などがある.

```
1 for i in range(10):
2  print('Hello', i)
```



1 行目の意味は, 「変数 i を 0 から 10 未満の間で, 1 ずつ増やしながら繰り返す」
2 行目の意味は, 「Hello」という文字列と, 今の数字を表す「i」を出力.
繰り返すコードは, 1 マス開けて書くということに注意.

```
1 i=0
2 while i<10:
3  print('Hello', i)
4  i=i+1
```

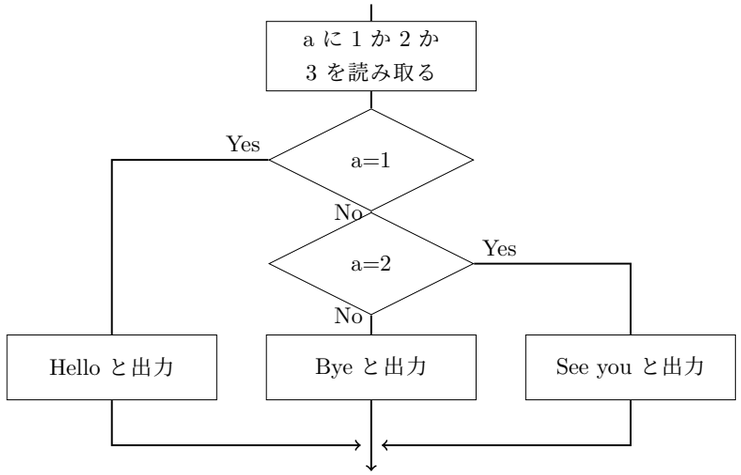


1 行目で i の初期値設定, 4 行目 (while 文内) で i の処理が必要.
2 行目の意味は, 「変数 i が 10 未満であれば繰り返す」
3 行目の意味は, 「Hello」という文字列と, 今の数字を表す「i」を出力.
繰り返すコードは, 1 マス開けて書くということに注意.

1.4　条件分岐

　条件によって実行内容を変えるときに利用.
1 行目は, 変数 a に入力された数字を代入. 入力された文字・数字は全て「文字列」として扱われるため, 「整数」に変換する必要がある. その関数として int を用いている.
2 行目は, もし a が 1 だったら...
4 行目は, そうでなくもし a が 2 だったら...
6 行目は, そうでなければ...
それぞれの条件下で実行するプログラムについては, 字下げを行う.

```
1 a=int(input('INPUT 1 or 2 or 3: '))
2 if a==1:
3  print('Hello')
4 elif a==2:
5  print('See you')
6 else:
7  print('Bye')
```



1.5　練習問題

(1) 自分の名前を出力させよ.

(2) a, b に好きな数字を当てはめ, 全ての四則演算を実行し, 結果を表示させよ.

(3) 0 から 10 回繰り返し, Hello と 今の番号の 2 倍の数を表示させよ.

(4) 1 ～ 4 のいずれかを実行後に入力し, 読み取った数字によって違う英単語を出力させるプログラムを作れ.

(5) 自由にプログラムを組み, いろいろと試してみる.

2　リストを使う

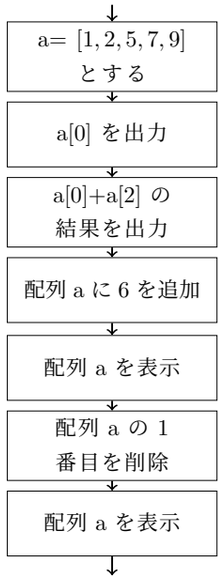
評価	C	B	A
基準	B に満たない.	リストについて理解し, 例文を正しく実行することができる.	練習問題を, 例題を参考に正しく実行することができる.

2.1　使い方

リストを用いることで, いくつかの数字をまとめて扱うことができる.

注) リストの先頭は 0 番目.

```
1 a=[1, 2, 5, 7, 9]
2 print(a[0])
3 print(a[0]+a[3])
4 a.append(6)
5 print(a)
6 a.pop(1)
7 print(a)
```



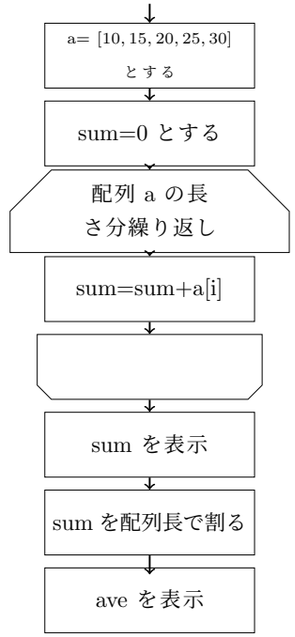
1 行目でリストの定義, 2-3 行目はリストの要素の扱い方である.

4 行目で, リストの最後に 6 を追加.

6 行目で, リストの 1 番目の数字を削除.

2.2　合計・平均を求める

```
1 a=[10, 15, 20, 25, 30]
2 sum=0
3 for i in range(len(a)):
4  sum=sum+a[i]
5 print('SUMARRY', sum)
6 ave=sum/len(a)
7 print('AVERAGE', ave)
```



2 行目で, 合計を計算する変数の初期化.

3 行目の len という関数は, 配列の長さを求めてくれる関数である. この場合は, len(a)= 5 である.

2.3　練習問題

a= $[1, 3, 5, 7, 9, 11, 13, 15, 17]$ とする.

以下の問いには, 授業で学んだことを活用すること.

(1) 合計, 平均 をそれぞれ出力せよ.

(2) 分散 (V), 標準偏差 (σ) を求めよ.

$$V = \frac{1}{n}\sum_{i=1}^{n}(a_n - \overline{a})^2, \quad \sigma = \sqrt{V}$$

(3) b= $[2, 3, 2, 5, 8, 4, 10, 15, 14]$ とし, a と b の相関係数 (r) を求めよ.

$$r = \frac{cov(a,b)}{\sigma(a)\sigma(b)}, \qquad cov(a,b) = \frac{1}{n}\sum_{i=1}^{n}(a_i - \overline{a})(b_i - \overline{b})$$

3　乱数・関数

評価	C	B	A
基準	B に満たない.	各プログラムについて構造も理解し, 例文を正しく実行することができる.	目的のプログラムのためには, どのような構造のアルゴリズムが必要か検討することができる.

3.1　乱数を使う

3.1.1　コマンド

```
1 import random
2 a=random.randint(3, 6)
3 print(a)
4 b=random.randrange(8)
5 print(b)
```

1 行目は, 乱数機能を追加するコマンド.
2 行目は, 3 以上 6 以下のランダムな整数値を与える関数.
4 行目は, 0 以上 8 未満のランダムな整数値を与える関数.
与えられる乱数値は, 実行のたびに変化する.

3.1.2　ジャンケン (vs. CPU)

ランダムに手を出すコンピュータと, ジャンケンを行うプログラムを組んでみる.

```
1 import random
2 print("Your hand? 1:Rock, 2:Scissors, 3:Paper")
3 user_hand=int(input("INPUT 1 or 2 or 3 : ", ))
4 cpu_hand=random.randint(1,3)
5 if cpu_hand==1:
6  print("CPU is Rock")
7 elif cpu_hand==2:
8  print("CPU is Scissors")
9 else:
10 print("CPU is Paper")
```

(1) ジャンケンプログラムのフローチャート図を描いて, 構造を理解しよう.

(2) 実際に動かしてみよう.

3.2 関数

3.2.1 プログラミング言語における関数とは

同じ機能を持つプログラムを何度も書くと, コードが膨大になり読みにくい. それを避けるために, 関数を作成し整理することができる.

```
1 def circle(r):
2  return r**2*3.14
3
4 r=int(input('radius='))
5 area=circle(r)
6 print("AREA=", area)
```

数学の関数同じように, $f(x)$ の形で表す.
f を関数名, x を引数, x を入れると返ってくる値を返り値 (戻り値) という. 上の例では, circle が関数名, r が引数, $3.14 \times r^2$ が返り値である.

3.3 その他関数

- 組み込み関数
  あらかじめ用意されている関数のこと. python には, 豊富に用意されている.
- API
  プログラムからソフトウェアを操作するためのインタフェース.
- WebAPI
  外部から呼び出して利用できる API のこと. 必要なデータを引数として送ることで, 返り値としてサービス・データが利用できる.

3.4 演習問題

例で挙げたコードを改良し, ジャンケンの勝敗も表示させるプログラムを作りたい.

(1) どのような構造で作成すればよいだろうか. 下のスペースにフローチャート図を描いてみよう.

(2) 実際にコードを書いて, 実行してみよう.

4　コンピュータの計算時間

評価	C	B	A
基準	B に満たない.	例文を正しく実行することができる. コンピュータの計算時間について理解し評価できる.	与えられたプログラムの計算時間を改良することができる.

一般に「計算量」というと, 「時間計算量」と「空間計算量」の 2 つがあるが, これ以降「計算量」というと「時間計算量」のこととする.

時間計算量	実行時間に関する計算量のこと.
空間計算量	メモリ使用量に関する計算量のこと. 変数, 配列, ポインタ, スタックなどのデータ構造を使用する際に必要となるメモリの量.

4.1　時間計算量

アルゴリズムの評価の 1 つで, 入力サイズ n の増加に対して実行時間はどれくらいの割合で増加するかの指標. 例えば 2 つのアルゴリズム A と B に対し...

$$\begin{array}{cc} \text{入力サイズ} & 1, 2, 3, \cdots, n \\ \text{A} & 1, 2, 3, \cdots, n \\ \text{B} & 1, 4, 9, \cdots, n^2 \end{array}$$

という計算量のとき, アルゴリズム A の方が性能がいいことがわかる. A の計算量を $O(n)$, B の計算量を $O(n^2)$ と書く.

【計算量の求め方とルール】これを計算量として表示する際は,

```
1  i=0
2  while i<N:
3   print("Hello")
4   i=i+1
5
6  j=0
7  k=0
8  while j<N:
9   while k<N:
10   print("Hello")
11   k=k+1
12  j=j+1
```

1 行目の, $i = 0$ という動作で計算量 1 回.
2〜4 行目は, N 回分の「Hello と表示」「$k = k + 1$」という操作があるので, 計算量 $2N$ 回.
6, 7 行目の, $j = 0, k = 0$ という操作で計算量 2 回.
8 行目の while 文の中に, 9〜11 行目の繰り返し構造がある. 各 j に対し $2N$ 回ずつあり, また「$j = j + 1$」という操作もあるため, 計算量は $(2N^2 + N)$ 回.
よって, このコードの総計算量は,

$$1 + 2N + 2 + 2N^2 + N = 2N^2 + 3N + 3 \text{ 回}$$

である.

(1) 増加率が一番大きいもののみ考える.
(2) 係数は無視して考える.

というルールがあるため, 計算量は $O(n^2)$ として表す.

【表記の仕方とアルゴリズム例】

(1) $O(1)$

```
1  print("Hello")
2  print("See You")
```

(2) $O(n)$

```
1  i=0
2  while i<N:
3   print("Hello")
4   i=i+1
```

(3) $O(n^2)$

```
1  while i<N:
2   while j<N:
3    print("Hello")
4    j=j+1
5   i=i+1
```

(4) $O(\log n)$
対数関数に比例して, 計算量が増加するということである. 例えば, A さんの年齢 $(0 \sim 100)$ を聞く際に,

$$1 \text{ 歳ですか. } 2 \text{ 歳ですか. } 3 \text{ 歳ですか. } \cdots$$

だと, かかる時間は $O(n)$ である (これを線形探索という). しかし,

$$50 \text{ 歳より上ですか. } \cdots$$

のように, 半分ずつに絞っていけば早く終わる. このような探索手法を「二分木探索」という. $O(logn)$ の計算量で済むアルゴリズムの代表的な例である.

〈感覚的な説明〉
半分ずつに分割するので, N 個の探索に 2^n 回で到達できるとする.

$$\begin{aligned} 2^n &= N \\ \log_2 2^n &= \log_2 N \\ n &= \log_2 N \end{aligned}$$

ということで, n は対数関数に比例していることがわかる.
本来は, $\log n$ の省略されている底の部分は, 自然対数

$$e = \lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n$$

であるが, 詳しくは数学で.

4.2 例題

入力された数字が素数か否か判定するプログラムを以下に示す.

```
print("Please input the number.")
N=int(input())
i=2
A=1
while i < N:
 if N%i==0:
  print(N, "is not a prime number.")
  A=0
  break
 i=i+1
if A==1:
 print(N, "is a prime number.")
```

(1) フローチャート図を描いてみよう.

(2) 計算量を求めよう.

(3) 実際に実行してみよう. いろいろな数字で試してみよう.

4.3 練習問題

素数判定をもっと早くできないか検討してみよう.
(条件を加えても可. 例: 100 以下に限定する)

5　さまざまなプログラム

5.1　演習 1 日目

評価	C	B	A
基準	B に満たない.	与えられたプログラムについて構造を理解し, 正しく実行することができる.	練習問題を, 例題を参考に正しく実行することができる.

```
1 N=int(input("Input a number ",))
2 for i in range(N):
3  for j in range(N-i):
4   print(" ",end="")
5  for j in range(2*i+1):
6   print("*",end="")
7  print("")
```

注) プリント文の「,end=””」の部分は, 強制的な改行キャンセル.
また, 7 行目に「 print(””)」を書いたのは, このタイミングで改行するため.

問題
上記のプログラムを見て, 以下の問いに答えよ.

(1) 入力する数字を変えて, 表示されるものを確認せよ.

(2) フローチャート図で, 構造を確認せよ.

(3) 以下のようなプログラムを作れ.

(a) 入力した数字の段数の逆三角形

```
1  *******
2   *****
3    ***
4     *
```

(b) 入力した数字が太さとなる十字架 (例は $n = 3$ のとき)

```
1     ***
2     ***
3     ***
4  *********
5  *********
6  *********
7     ***
8     ***
9     ***
```

(c) 自身の好きな図形に対し, 入力される数字に応じてサイズが変わるように作れ.

5.2　演習 2 日目

評価	C	B	A
基準	B に満たない.	簡易的なコードから, 作りたいプログラムの構造組み立てができる.	構造組み立ての結果から, 例題を正しく実行することができる.

サマーウォーズにて, 健二が生年月日から曜日を暗算で求めていた. 彼は「モジュロ演算」をしただけである. このモジュロ演算を実際に学び, コードを組んでみよう.

(1) ツェラーの方法 (簡略化 ver.)

(a) 1 月, 2 月の場合, その前の年の 13 月, 14 月として扱う.

(b) A = 西暦の下 2 桁 とする.

(c) B = 西暦の下 2 桁を 4 で割った商

(d) 何月かにより, 以下の表に対応する値を C とする.

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

(e) D = 日付 とする.

(f) $X = A + B + C + D$ を求める. もし, 1900 年台の場合は 1 を足す.

(g) X を 7 で割ったあまりを求める. 以下の表に対応して曜日を求めることができる.

余り	1	2	3	4	5	6	0
曜日	日	月	火	水	木	金	土

(2) 擬似 (簡易) コード

(a) 生まれた年の入力

(b) 生まれた月の入力

(c) 生まれた日の入力

(d) もし, 1 月か 2 月なら年を −1 し, 13 月, 14 月とする.

(e) もし, 1 月か 2 月なら, 13 月, 14 月と置き換える.

(f) A, B, C, D をそれぞれ求める.

(g) A, B, C, D の和を求める.

(h) もし, 1900 年台であれば, 先の和に 1 を足す.

(i) 7 で割った余りを求め, その値によって曜日を決定する.

(3) 上の擬似 (簡易) コードを参考に, 曜日を求めるためのフローチャートを検討せよ.

(4) 曜日を求めるプログラムを組め.

5.3 答え (例)

```
year=int(input("Year= "))
month=int(input("Month= "))
day=int(input("Day= "))
if month==1:
 newyear=year-1
 month=13
elif month==2:
 newyear=year-1
 month=14
else:
 newyear=year
a=newyear%100
b=(newyear%100)//4
c_list=[3,6,1,4,6,2,5,0,3,5,1,4]
c=c_list[month-3]
d=day
x=a+b+c+d
if (newyear//100)==19:
 x=x+1
w=x%7
week=["Sataday","Sunday","Monday","Tuesday","Wednesday","Thursday","Friday"]
print(day,"/",month,"/",year,"is...")
print(week[w])
```

【工夫点】

- c を求める際に, 「if」文で条件分岐しても良かったが, 配列で扱う方がコードが冗長にならないので, そのようにした.

- 最後の曜日を求める際も同様.

5.4 演習 3 日目

評価	C	B	A
基準	B に満たない.	入力に対する結果を予想できる. 与えられたプログラムについて構造を理解し, 正しく実行することができる.	練習問題を, 例題を参考に正しく実行することができる.

```
a=[]
A0=int(input("First Term ",))
d=int(input("Common Deference ",))
N=int(input("number ",))
a.append(A0)
A=A0
for i in range(N-1):
 A=A+d
 a.append(A)
print("a= ",a)
sum=0
for i in range(len(a)):
 sum=sum+a[i]
print("SUM= ",sum)
```

注) 「a=[ ]」で, 要素のない配列生成.

問題

上記のプログラムを見て, 以下の問いに答えよ.

(1) $(a, d, N) = (3, 4, 5)$ と入力した際, 何が表示されるでしょうか.

(2) これは, 何をするプログラムでしょうか.

(3) フローチャート図で, 構造を表せ.

(4) 初項, 公比, 項数を与えた際に, 同様の表示を行うプログラムを作れ.

5.4.1 練習問題

フィボナッチ数列について考える. ここで, フィボナッチ数列とは,

$$1,\quad 1,\quad 2,\quad 3,\quad 5,\quad 8,\quad 13,\quad \cdots$$

のように, $a_n = a_{n-1} + a_{n-2}$ $(n \geqq 3)$ で定義される数列である.

(1) 実行後の入力 N に対し, この数列の第 N 項までを出力するプログラムを作れ.

(2) また, 第 N 項までの和を求めるプログラムを作れ.

5.4.2 解答例

フィボナッチ数列の解答例

```
a=[1,1]
n=int(input("number = "))
for i in range(n-2):
 a.append(a[i+1]+a[i])
print(a)

sum=0
for i in range(len(a)):
 sum=sum+a[i]
print("SUM = ",sum)
```

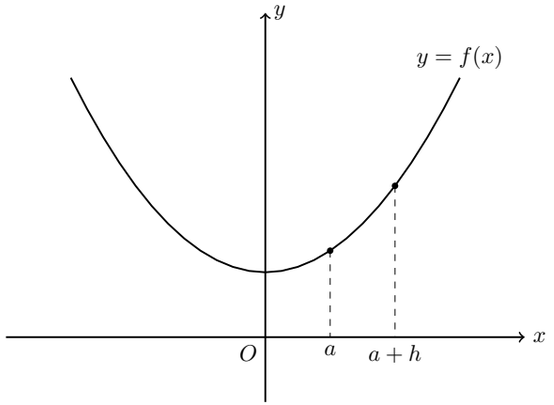
5.5 演習 4 日目

評価	C	B	A
基準	B に満たない.	与えられたプログラムを理解し, 正しく実行することができる.	プログラムの構造を理解し, 数学で学んだこととの関連付けができる.

5.5.1 微分

数学の復習

微分係数の確認.



$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{(a+h) - a}$$

```
1  def f(x):
2   y=x**2
3   return y
4
5  a=int(input("a= "))
6  for i in range(5):
7   h=1/(10**i)
8   f_prime=(f(a+h)-f(a))/h
9   print("small disatnce ",h)
10  print(" -> ",f_prime)
```

問題

上記のプログラムを見て, 以下の問いに答えよ.

(1) 実行後, 1 と入力した際, 何が表示されるでしょうか.

(2) 実際の微分係数 $f'(1)$ に近づいているか検証せよ.

(3) 関数をいろいろ変えて実験せよ.

(a) $f(x) = x^2 + 3$

$h =$	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$a = 3$					
$a = -2$					

(b) $f(x) = x^3 + 3x^2 + 4$

$h =$	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$a = 0$					
$a = 2$					

(c) $f(x) = 2^x$

$h =$	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$a = 2$					
$a = -3$					

(d) $f(x) = x^x$

$h =$	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$a = 2$					
$a = -1$					

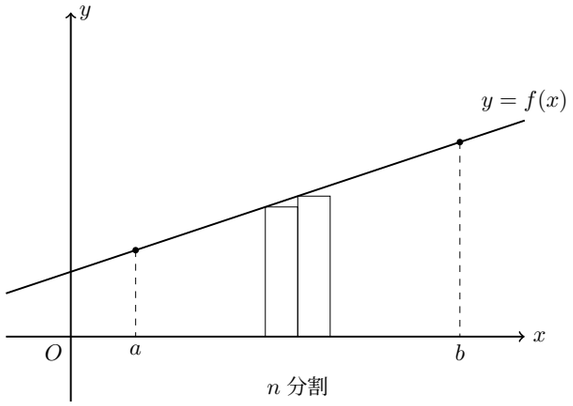
(e) $f(x) = \sqrt{x}$

$h =$	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$a = 3$					
$a = 1$					

5.5.2 積分

復習

数学の確認 (区分求積法)



n 分割

$$\int_a^b f(x)dx = \lim_{n\to\infty}\sum_{k=0}^{n}\frac{1}{n}f\left(a+\frac{k}{n}(b-a)\right)$$

```
def f(x):
 y=x
 return y

a=int(input("a= ",))
b=int(input("b= ",))
for i in range(6):
 h=(b-a)/(10**i)
 sum=0
 x=a
 for i in range(10**i):
  sum=sum+h*f(x)
  x=x+h
 print("small disatnce ",h)
 print(" -> ",sum)
```

問題

上記のプログラムを見て, 以下の問いに答えよ.

(1) 実行後, $a=1, b=3$ と入力した際, 何が表示されるでしょうか.

(2) 実際の定積分の値 $\int_1^3 xdx$ に近づいているか検証せよ.

(3) 関数をいろいろ変えて実験せよ.

(a) $f(x)=x^2+3$

$h=$	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$(a,b)=(1,2)$					
$(a,b)=(-1,2)$					

(b) $f(x)=x^3+3x^2+4$

$h=$	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$(a,b)=(0,1)$					
$(a,b)=(1,3)$					

(c) $f(x)=2^x$

$h=$	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$(a,b)=(0,1)$					
$(a,b)=(1,3)$					

(d) $f(x)=x^x$

$h=$	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$(a,b)=(0,1)$					
$(a,b)=(2,3)$					

(e) $f(x)=\sqrt{x}$

$h=$	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$(a,b)=(0,1)$					
$(a,b)=(1,3)$					

6　シミュレーション

評価	C	B	A
基準	B に満たない.	シミュレーションを理解し, 正しく実行することができる.	練習問題を, 例題を参考に正しく実行することができる.

6.1　コイン投げ

表裏が同じ確率で出るサイコロについて, 考える. 何回投げれば表の出た確率が $\frac{1}{2}$ に収束するか.

```
import random
N=int(input("how many coin",))
count=0
for i in range(N):
 coin=random.randrange(2)
 if coin==0:
  count=count+1
print("probably of omote =",count/N)
```

【実験結果】
コイン投げの回数を変えて各々 4 回ずつ実験しよう.

	10 回	100 回	1000 回	10000 回	100000 回
1 回目					
2 回目					
3 回目					
4 回目					
平均					

(参考) 大数の法則　law of large numbers

表の出る確率が $p=\frac{1}{2}$ であるコインについて, 投げた回数を n, 表の出た回数を r とする. このときの表の出た確率 $\hat{p}=\frac{r}{n}$ について

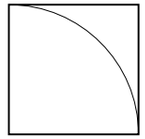
$$\lim_{n\to\infty}\hat{p}=p$$

が成立. (つまり, 投げる回数増やせば出た確率は p に収束していくということ.)

6.2　モンテカルロ法

モンテカルロ法によって, 円周率の近似値を求めてみよう. 以下の手順で求めていく.

(1) 右図のように, 1 辺が 1 の正方形の中に $\frac{1}{4}$ 円を考える.
(2) 0 以上 1 以下の小数乱数で, (x,y) 座標を生成する.
(3) この座標が扇形の中に入っている $(x^2+y^2\leqq 1)$ か否かを評価する.
(4) (2),(3) を繰り返す.
(5) (扇形内の個数)÷(点を打った個数) の値を 4 倍すると, π の近似値を出せる.



(参考) モンテカルロ法による円周率近似

右上図における扇形の面積は,

$$\pi\times 1\times 1\times\frac{1}{4}=\frac{1}{4}\pi$$

であり, 正方形の面積は 1 である. つまり, 正方形の中にランダムに点を打っていくと,

$$(\text{扇形内の点}):(\text{全体})=\frac{1}{4}\pi:1$$

になっていくはず. よって, (扇形内の点) ÷ (全体) $\times 4\to\pi$　$(n\to\infty)$

```
import random
N=int(input("how many point ",))
count=0
for i in range(N):
 x=random.uniform(0,1)
 y=random.uniform(0,1)
 if x**2+y**2<=1:
  count=count+1
print(count*4/N)
```

注) 「random.uniform(0,1)」で, 0 以上 1 以下の小数乱数生成.
【実験結果】

$N=$	10 回	100 回	1000 回	10000 回	100000 回
1 回目					
2 回目					
3 回目					
平均					

6.3 練習問題 1

コイン投げを参考に, サイコロのシミュレーションを行なってみよう. (表の中には確率を記入)
(ヒント：配列を活用)

(1) $N = 100$

出た目	1	2	3	4	5	6
1 回目						
2 回目						
3 回目						
平均						

(2) $N = 1000$

出た目	1	2	3	4	5	6
1 回目						
2 回目						
3 回目						
平均						

(3) $N = 10000$

出た目	1	2	3	4	5	6
1 回目						
2 回目						
3 回目						
平均						

(4) この実験での 1 回目, 2 回目, ... も, 同一プログラム内で実行可能である. やってみよう.

(5) サイコロの出る目が同様に確からしくない場合にどうなるだろうか. 検証してみよう.

6.3.1 模範解答例

サイコロの目の実験において, N 回のシミュレーションを M 回分行い, 平均回数とその比率を自動で計算してくれるアルゴリズムを組んだ.

```
import random
COUNT=[]
count=[0,0,0,0,0,0]
N=100000
M=4
for i in range(M):
 count=[0,0,0,0,0,0]
 for j in range(N):
  me=random.randrange(6)
  count[me]=count[me]+1
 COUNT.append(count)
for i in range(len(COUNT)):
 print("No.",i," : ",COUNT[i])
AVERAGE=[0,0,0,0,0,0]
for i in range(len(COUNT)):
 for j in range(len(COUNT[i])):
  AVERAGE[j]=AVERAGE[j]+COUNT[i][j]
for i in range(len(AVERAGE)):
 AVERAGE[i]=AVERAGE[i]/M
print("AVE = ",AVERAGE)
for i in range(len(AVERAGE)):
 AVERAGE[i]=AVERAGE[i]/N
print("PLO = ",AVERAGE)
```

7　ヌメロン

評価	C	B	A
基準	B に満たない.	与えられたプログラムについて構造を理解し, 正しく実行することができる.	練習問題を, 例題を参考に正しく実行することができる.

ヌメロンというゲームを知っているだろうか. 各自 3 桁の数字を作成し, 相手の数字を当てるゲームである. 3 桁といったが, 0 から始めてもいいが, 同じ数字の重複は許されない.
【ルール】

(1) 先攻・後攻を決定.
(2) 先攻は相手の数字を予想し, その数字をコールする.
(3) 後攻は, コールされた数字と自身の数字がどの程度合っているかを発表する.
　(a) 数字と桁が揃っている場合「EAT」.
　(b) 数字は合っているが, 桁が違う場合「BITE」
　　コール (135), 自身の番号 (156) の場合, 「1-EAT, 1-BITE」
(4) (2),(3) を先攻・後攻交互に繰り返し, 先に相手の番号を完全に当てた方が勝ちである.

これを, 授業用に少しルール変更してプログラミングしてみた.

【授業用ヌメロンルール】

(1) コンピュータの数字を 10 回以内に当てることができれば勝ち.
(2) 人間はコンピュータの数字を予想し, その数字をコールする.
(3) 以下の場合はその CALL はスキップ.
　(a) 入力が 0 ～ 999 の間ではない.
　(b) 各桁の中に同じ数字が入っている.
(4) コンピュータはコールされた数字がどの程度合っているかを発表する.

7.1　練習

(1) とりあえず実行してみる.
(2) 右のコードの中で, 各部分が何をしているのかの構造分解をしてみる.

注) if 文の条件について, 「!=」で, 「not equal」. 「or」で「または」.「and」で「かつ」
空白を入れているのは, 見やすくするため.

7.2　ヌメロンコード (授業用)

```
import random
print("Let's play numeron.")
print("You have ten times of chances.")
print(" ")
i=0
cpu_num1=random.randrange(10)
while i==0:
 cpu_num2=random.randrange(10)
 if cpu_num2!=cpu_num1:
  break
while i==0:
 cpu_num3=random.randrange(10)
 if (cpu_num3!=cpu_num1)and(cpu_num3!=cpu_num2):
  break
cpu_num=[cpu_num1,cpu_num2,cpu_num3]
for i in range(10):
 print("No.",i+1)
 n=int(input("CALL = ",))
 call=[0,0,0]
 call[0]=n//100
 call[1]=(n-call[0]*100)//10
 call[2]=(n-call[0]*100-call[1]*10)
 if (call[0]==call[1])or(call[1]==call[2])or(call[2]==call[0]):
  print("Fraud. Please change the number of each figure.")
  print("Pass")
 elif not(9<n<1000):
  print("Fraud. Please input three columns of numbers.")
  print("Pass")
 else:
  bite=0;eat=0
  for j in range(3):
   if call[j]==cpu_num[j]:
    eat=eat+1
  for j in range(3):
   for k in range(3):
    if (j!=k)and(call[j]==cpu_num[k]):
     bite=bite+1
  print(" ",eat,"- EAT")
  print(" ",bite,"- BITE")
  if eat==3:
   print("You Win")
   break
 if i==9:
  print("Yow Lose")
  print("CPU's number is",cpu_num)
 print(" ")
print("Game End")
```

7.3 演習

演習用のファイルで作成すること.

(1) チャンスの回数も自身で入力できるようにしてみる.

(2) CALL の際に数値以外を入力するとエラーを起こしてしまう. 例文と同様にスキップするためにはどうしたらいいだろうか.

〈ヒント 1〉

以下の関数を定義することで, s が int 型 (整数) に変換できるか否かを確認. できなかった場合は False を返す.

```
def isint(s):
 try:
  int(s)
 except ValueError:
  return False
 else:
  return True
```

〈ヒント 2〉

以下の確認を入れて, 条件を満たせばスキップにし, それ以外の場合には数値に変換し, 数値の比較検証に移ればいい.

```
if isint(n)==False:
```

7.3.1 解答例

チャンスの回数については,

```
N=int(input("chance = "))
```

とでも書いて, 繰り返し回数を N にすればいい.

8　探索

評価	C	B	A
基準	B に満たない.	探索アルゴリズムについて理解できる.	正しく実行することができる.

配列の中から, 探している値が何番目にあるのかを探すアルゴリズムを考える.

8.1　線形探索

純粋に左から順に 1 個ずつ調べていく方法.

手順書

配列 a と探している数字が与えられたとき, 以下の手順で探索する.

(1) a[0] が探している数字と同じか調べる.
　(a) もし同じであれば, 0 を返し, 繰り返しを抜ける.
(2) この操作を残り $1, 2, \cdots$ で配列の長さ分だけ繰り返す.
(3) 繰り返した回数が配列の長さになったときに見つかっていない場合は, 探している値は存在しない.

8.1.1　仕組みを捉える

トランプを用いて, 上の手順通りに探索してみよう.

8.1.2　python で線形探索

[ ] に当てはまる文を考えて入れよう.
(配列の中の要素は各自好きに入れると良い.)

```
a=[1, 3, 2, 5, 12, 4, 7, 8, 21, 10, 9]
search_num=int(input("search number = ", ))
for i in [ ]:
 if a[i]==search_num:
  print("No. ",i)
  break
 [ ]:
  print("Not found.")
```

8.2　二分岐探索

事前に昇順に並んだデータに対して使える探索手法. 目的の値が中央よりも右か左かを探索していく. 計算量は $O(\log n)$ であり, かなり効率のいいアルゴリズム.

手順書

昇順に並んだ配列 a と探している数字が与えられたとき, 以下の手順で探索する.

(1) 探索範囲の左端の配列番号を left , 右端の配列番号を right で初期化する.
(2) left の値が right の値以下の場合, 以下を繰り返す.
　(a) (left+right)÷2 の商の値を mid とする.　(中央の値を得る)
　(b) mid 番目の配列が調べている数であれば, その配列番号を出力してループ脱出.
　(c) mid 番目の配列 < 調べている数であれば, left を mid+1 にする.
　(d) mid 番目の配列 > 調べている数であれば, right を mid−1 にする.
(3) left>right の場合, Not Found と出力する.

8.2.1　仕組みを捉える

トランプを用いて, 上の手順通りに探索してみよう.

8.2.2　python で二分岐探索

[ ] に当てはまる文を考えて入れよう.
(配列の中の要素は各自好きに入れると良い.)

```
data=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
search_num=int(input("search number=",))
left=0
right=len(data)-1
while [ ]
 mid=[ ]
 if data[mid]==search_num:
  print("No.",mid)
  break
 elif [ ]
  left = mid + 1
 else:
  right = mid - 1
if left>right:
 print("Not found")
```

8.2.3 模範解答例

線形探索

```
a=[1, 3, 2, 5, 12, 4, 7, 8, 21, 10, 9]
search_num=int(input("search number = ", ))
for i in range(len(a)):
 if a[i]==search_num:
  print("No. ",i)
  break
 if i==len(a)-1:
  print("Not found.")
```

二分岐探索

```
data=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
search_num=int(input("search number=",))
left=0
right=len(data)-1
while left<=right:
 mid=(left+right)//2
 if data[mid]==search_num:
  print("No.",mid)
  break
 elif data[mid]<search_num:
  left = mid + 1
 else:
  right = mid - 1
if left>right:
 print("Not found")
```

9　ソートアルゴリズム

評価	C	B	A
基準	B に満たない.	ソートアルゴリズム構造を理解し, 正しく実行することができる.	様々なソートアルゴリズムの構造を学び, 理解する.

9.1　バブルソート

手順書

配列 a を以下の手順で並べ替えていく.

(1) 以下の操作を $i = 0$ から (配列長 -1) 回繰り返す.
　(a) 以下の操作を $j = 0$ から ((配列長) $- i - 1$) 回繰り返す.
　　i. j 番目の数 $> j + 1$ 番目の数であれば数の入れ替え
　　ii. そうでなければ何もしない.

9.1.1　仕組みを捉える

トランプを用いて, 上の手順書通りに並べ替えてみよう. 何手で並べ替えることができるか. 入れ替わりの流れを残しておこう.

9.1.2　python でバブルソート

```
l=[2,3,1,5,9,6,7]
for i in range(len(l)-1):
 for j in range(len(l)-i-1):
  if l[j]>l[j+1]:
   tmp=l[j+1]
   l[j+1]=l[j]
   l[j]=tmp
print(l)
```

9.2　選択ソート

手順書

配列 a を以下の手順で並べ替えていく.

(1) 以下を (配列長 $-$ 1) 回繰り返す.
　(a) least=i とする.
　(b) 以下を $j = i + 1$ から (配列長 $-$ 1) 未満で繰り返す.
　　i. 配列の j 番目の数と least 番目の数を比較し, j 番目が小さければ least=j とする.
(2) i 番目の配列と least 番目の配列を入れ替える.

9.2.1　仕組みを捉える

トランプを用いて, 上の手順書通りに並べ替えてみよう. 何手で並べ替えることができるか. 入れ替わりの流れを残しておこう.

9.2.2　python で選択ソート

```
l=[1,6,3,9,2,7,5]
for i in range(len(l)-1):
 least=i
 for j in range(i+1,len(l)):
  if l[j]<l[least]:
   least=j
 tmp=l[i]
 l[i]=l[least]
 l[least]=tmp
 print(l)
```

9.3 その他のソート

授業では扱わないが, 他にも様々なソートの手法がある.

(1) 挿入ソート
(2) クイックソート
(3) マージソート

などなど...

これらの手法の方法を調べ, 自分なりにまとめてみよう. (手順書 + ソート例)

10　2 進数, 10 進数, 16 進数の変換

評価	C	B	A
基準	B に満たない.	与えられたプログラムについて構造を理解し, 正しく実行することができる.	練習問題を, 例題を参考に正しく実行することができる.

10.1　復習

以下の数を [ ] に書かれた進数へ変換せよ.

(1) 135_{10} [2 進数]

(2) 4024_{10} [16 進数]

(3) 100111_{2} [10 進数]

(4) $d2a_{16}$ [10 進数]

(5) fa_{16} [2 進数]

10.2　10 進数から 2 進数

入力された数 n に対して, 以下の手順書で変換してみる.

手順書

(1) 空の配列 a を作る.
(2) 以下を $max = 0$ から $2^{max} < n$ を満たす間繰り返す.
　(a) $max = max + 1$
(3) 以下を $i = 0$ から $max + 1$ 回繰り返す.
　(a) $n \geqq 2^{max-i}$ の場合, 次の 2 つの操作を行う.
　　i. $n = n - 2^{max-i}$
　　ii. 配列 a に 1 を追加
　(b) そうでない場合, 以下の操作を行う.
　　i. 配列 a に 0 を追加
(4) 配列 a を出力.

この手順がどのように動いているか, $n = 17$ に対して構造分析してみよう.

10.2.1　python で変換

[ ] に当てはまる文を入れて実行しよう. (2 行目の [] は, 空配列作成なのでそのまま.)

```
1  n=int(input("number="))
2  a=[]
3  max=0
4  while (2**max < n):
5   max=max+1
6  for i in range(max+1):
7   [ ]
8    [ ]
9    [ ]
10  else:
11   a.append(0)
12 print(a)
```

10.3　2 進数から 10 進数

手順書

入力された数 binary に対し, 以下の操作を行う.

(1) 空の配列 a の作成, n の初期化 $(=0)$.
(2) binary$\geqq 10^n$ の間以下を繰り返す.
　(a) $n=n+1$
(3) i を n 回繰り返す.
　(a) 配列 a に, binary を (10^{n-i-1}) で割った商を追加.
　(b) binary を 「binary を 10^{n-i-1} で割ったあまり」で更新.
(4) sum の初期化 $(=0)$.
(5) i を配列 a の長さ分繰り返し.
　(a) sum に $a[i]\times 2^{len(a)-i-1}$ を足す.
(6) sum の表示.

この手順で変換ができるのかを確かめよう. また, なぜできるのか検討しよう.

10.3.1　python で変換

[ ] に当てはまる文を入れて実行しよう. (2 行目の [] は, 空配列作成なのでそのまま.)

```
1  binary=int(input("binary number="))
2  a=[]
3  n=0
4  while binary>=10**n:
5   n=n+1
6  for i in range(n):
7   [ ]
8   [ ]
9  sum=0
10 for i in range(len(a)):
11  [ ]
12 print(sum)
```

10.4　練習

他の変換も挑戦しよう. (自身の計算手順を参考に !)

(1) 10 進数 $\longrightarrow$16 進数

(2) 2 進数 $\longrightarrow$16 進数

考慮すべき事項 (アイデア)

- a, b, c, d, e, f が数字として扱えない. 10, 11, 12, 13, 14, 15 として扱う ?
- 2 進数から 16 進数は間に 10 進数を挟むといい ?

10.4.1 模範解答例

10 進数から 2 進数

```
n=int(input("number="))
a=[]
max=0
while (2**max < n):
 max=max+1
for i in range(max+1):
 if n>=2**(max-i):
  n=n-2**(max-i)
  a.append(1)
 else:
  a.append(0)
print(a)
```

2 進数から 10 進数

```
binary=int(input("binary number="))
a=[]
n=0
while binary>=10**n:
 n=n+1
for i in range(n):
 a.append(binary//(10**(n-i-1)))
 binary=binary%((10**(n-i-1)))
sum=0
for i in range(len(a)):
 sum=sum+a[i]*2*(len(a)-i-1)
print(sum)
```

11　日常生活とプログラミング

11.1　公的年金等控除

評価	C	B	A
基準	B に満たない.	仕組みについて理解し, 入力された数に対する実行結果を自身で計算できる.	正しくプログラムし, 実行することができる.

11.1.1　言葉の意味

- 控除とは
  ··· 税金を計算する際に, 一定額を収入から差し引くこと. その目的は, 最低限の生活を保証するためとされている.
- 公的年金とは
  ··· 国が運営する年金のこと. 現在では「国民年金」「厚生年金」の 2 種類がある.

11.1.2　公的年金控除

公的年金等の受給者の年齢 (その年の 12 月 31 日時点) と, 公的年金等の収入額に応じて, 以下の表の通りに公的年金控除が決められている.

受給者年齢	公的年金等の収入金額	公的年金等控除額
65 歳未満	130 万円以下	60 万円
	130 万超　410 万円以下	収入金額 ×25%+27 万 5 千円
	410 万超　770 万円以下	収入金額 ×15%+68 万 5 千円
	770 万超　1000 万円以下	収入金額 ×5%+145 万 5 千円
	1000 万円超	195 万 5 千円
65 歳以上	130 万円以下	110 万円
	130 万超　410 万円以下	収入金額 ×25%+27 万 5 千円
	410 万超　770 万円以下	収入金額 ×15%+68 万 5 千円
	770 万超　1000 万円以下	収入金額 ×5%+145 万 5 千円
	1000 万円超	195 万 5 千円

11.1.3　手計算

以下の条件下での今年 (2023 年) の公的年金等控除額を求めよ.

(1) 生年月日：1959 年 12 月 30 日, 公的年金収入額：120 万円

(2) 生年月日：1950 年 12 月 20 日, 公的年金収入額：200 万円

(3) 生年月日：1955 年 7 月 12 日, 公的年金収入額：300 万円

(4) 生年月日：1960 年 10 月 15 日, 公的年金収入額：500 万円

(5) 生年月日：1958 年 1 月 1 日, 公的年金収入額：800 万円

(6) 生年月日：1951 年 7 月 10 日, 公的年金収入額：2000 万円

11.1.4 python で求めてみる

まず, 12 月 31 日時点での年を求める. その後, 控除額を計算.

手順書

(1) 今日の年を, year として読み込む.
(2) ユーザの生まれた年を入力させ, user_year として読み込む.
(3) age=year−user_year とする.
(4) 公的年金等の収入額を入力させ, income として読み込む.
(5) 以下, 条件分岐により控除額表示

以下は, 手順書の年齢を求めるまでのコードである. それ以降は条件分岐等を使いコードを書いていこう.

```
1 import datetime
2 dt_now=datetime.datetime.now()
3 year=dt_now.year
4 user_year=int(input("year of your birthday",))
5 age=year-user_year
```

補足) import datetime で, 今日の日付などを読み込む関数の取り込みを行う.

11.2 配偶者特別控除

評価	C	B	A
基準	B に満たない.	仕組みについて理解し, 入力された数に対する実行結果を自身で計算できる.	正しくプログラムし, 実行することができる.

11.2.1 言葉の意味

- 配偶者特別控除とは
  ··· 納税者本人と配偶者の合計所得金額に応じて以下の表の通り控除額を決定.

		納税者本人の合計所得金額		
		900 万円以下	900 万円超 950 万円以下	950 万円以上 1000 万円以下
配偶者合計所得金額	47 万円超 95 万円以下	38 万円	26 万円	13 万円
	95 万円超 100 万円以下	36 万円	24 万円	12 万円
	100 万円超 105 万円以下	31 万円	21 万円	11 万円
	105 万円超 110 万円以下	26 万円	18 万円	9 万円
	110 万円超 115 万円以下	21 万円	14 万円	7 万円
	115 万円超 120 万円以下	16 万円	11 万円	6 万円
	120 万円超 125 万円以下	11 万円	8 万円	4 万円
	125 万円超 130 万円以下	6 万円	4 万円	2 万円
	130 万円超 135 万円以下	3 万円	2 万円	1 万円

11.2.2 二次元配列

上記を場合分けして実装しようとすると, 条件分岐を多用する必要が出てくる. 二次元配列を使うと楽にできるので, それを導入しよう.

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

配列の中に配列を入れている状態である. 配列 a の 0 番目の配列の 1 番目は 2 であるが, それを以下のように表す.

```
a[0][1]
```

また, 配列 a の長さは 3 であるし, 配列 a の中の配列 a[0] は, 長さ 4 である.

11.2.3 二次元配列の練習問題

```
a=[[1,2,3,4],
[5,6,7,8],
[9,10,11,12]]
```

上の配列に対して, 以下の実行結果を求めよ.

(1)

```
print(a[2][1])
```

(2)

```
print(a[1])
```

(3)

```
sum=0
for i in range(len(a[1])):
 sum=sum+a[1][i]
print(sum)
```

(4)

```
for i in range(len(a[2])):
 a[2][i]=a[2][i]+a[1][i]
print(a)
```

11.2.4　python で実装

手順書

(1) 控除表の二次元配列, 所得の基準配列を作成する.
(2) 納税者, 配偶者の所得を入力させる.
(3) 対象外の人の排除.
(4) 納税者の所得区分を求める.
(5) 配偶者の所得区分を求める.
(6) 区分にあった控除額を表示.

[ ] に当てはまる文を補完して, 実装しよう.

```
deduction=[
[38,26,13],
[36,24,12],
[31,21,11],
[26,18,9],
[21,14,7],
[16,11,6],
[11,8,4],
[6,4,2],
[3,2,1]]
your=[900,950,1000]
spouse=[95,100,105,110,115,120,125,130,135]

your_income=int(input("your income= ",))
spouse_income=int(input("spouse income= ",))

if [ ]
 print("you are not eligible")
else:
 your_number=0
 spouse_number=0
 while your_income>your[your_number]:
  your_number=your_number+1
 [ ]
  [ ]
 print("Deduction amount= ",deduction[spouse_number][your_number])
```

11.2.5 模範解答例

公的年金控除

```
import datetime
dt_now=datetime.datetime.now()
year=dt_now.year
user_year=int(input("year of your birthday ",))
age=year-user_year
user_income=int(input("your income ",))
if user_income<=130:
 if age<65:
  print("koujo = 60")
 else:
  print("koujo = 110")
elif 130<user_income<=410:
 koujo=user_income*0.25+27.5
 print("koujo = ",koujo)
elif 410<user_income<=770:
 koujo=user_income*0.15+68.5
 print("koujo = ",koujo)
elif 770<user_income<=1000:
 koujo=user_income*0.5+145.5
 print("koujo = ",koujo)
else:
 print("koujo = 195.5")
```

配偶者控除

```
deduction=[
 [38,26,13],
 [36,24,12],
 [31,21,11],
 [26,18,9],
 [21,14,7],
 [16,11,6],
 [11,8,4],
 [6,4,2],
 [3,2,1]]
your=[900,950,1000]
spouse=[95,100,105,110,115,120,125,130,135]

your_income=int(input("your income= ",))
spouse_income=int(input("spouse income= ",))

if your_income>1000 or spouse_income<=47 or spouse_income>135:
 print("you are not eligible")
else:
 your_number=0
 spouse_number=0
 while your_income>your[your_number]:
  your_number=your_number+1
 while spouse_income>your[spouse_number]:
  spouse_number=spouse_number+1
 print("Deduction amount= ",deduction[spouse_number][your_number])
```