①令和４年 大学入試センター 試作問題

第３問 次の問い（**問１～３**）に答えよ。（配点　25）

問１　次の生徒（S）と先生（T）の会話文を読み，空欄 ア に当てはまる数字をマークせよ。また，空欄 イ ～ エ に入れるのに最も適当なものを，後の解答群のうちから一つずつ選べ。ただし，空欄 ウ ・ エ は解答の順序は問わない。

S：この前，お客さんが 460 円の商品を買うのに，510 円を払って，釣り銭を 50 円受け取っていたのを見て，授業で勉強したプログラミングで，そんな「上手な払い方」を計算するプログラムを作ってみたいと思いました。

T：いいですね。まず，「上手な払い方」とは何かを考える必要がありますね。

S：普通は手持ちの硬貨の枚数を少なくするような払い方でしょうか。

T：そうですね。ただ，ここでは，客が支払う枚数と釣り銭を受け取る枚数の合計を最小にする払い方を考えてみませんか？ 客も店も十分な枚数の硬貨を持っていると仮定しましょう。また，計算を簡単にするために，100 円以下の買い物とし，使う硬貨は 1 円玉，5 円玉，10 円玉，50 円玉，100 円玉のみで 500 円玉は使わない場合を考えてみましょう。例えば，46 円をちょうど支払う場合，支払う枚数はどうなりますか？

S：46 円を支払うには，10 円玉 4 枚，5 円玉 1 枚，1 円玉 1 枚という 6 枚で払い方が最小の枚数になります。

T：そうですね。一方，同じ 46 円を支払うのに，51 円を支払って釣り銭 5 円を受け取る払い方では，支払いに 2 枚，釣り銭に 1 枚で，合計 3 枚の硬貨のやり取りになります。こうすると交換する硬貨の枚数の合計が最小になりますね。

S：これが上手な払い方ですね。

T：そうです。このように，客と店が交換する硬貨の合計が最小となる枚数，すなわち「最小交換硬貨枚数」の計算を考えましょう。

S：どうやって考えればいいかなぁ。

T：ここでは，次の関数のプログラムを作り，それを使う方法を考えてみまし

使う硬貨
※100円以内の買い物

46円の支払い
10円4枚, 5円1枚, 1円1枚
⇒ 計6枚
51円支払い2枚, 5円釣り1枚
⇒ 計3枚

最小交換硬貨枚数
支払いと釣り銭とで用いる硬貨の合計が最小となる枚数

ょう。目標の金額を釣り銭無くちょうど支払うために必要な最小の硬貨枚数を求める関数です。

【関数の説明と例】

枚数(金額)… 引数として「金額」が与えられ，ちょうどその金額となる硬貨の組合せの中で，枚数が最小となる硬貨枚数が戻り値となる関数。
例：8 円は「5 円玉が 1 枚と 1 円玉が 3 枚」の組合せで最小の硬貨枚数になるので，**枚数**(8)の値は 4 となる。

T：これは，例えば，**枚数**(46)＝［ア］と計算してくれるような関数です。これを使って最小交換硬貨枚数の計算を考えてみましょう。例えば，46 円支払うのに，51 円払って 5 円の釣り銭を受け取る払い方をした場合，客と店の間で交換される硬貨枚数の合計は，この関数を使うと，どのように計算できますか？

S：［イ］で求められますね。

T：一般に，商品の価格 x 円に対して釣り銭 y 円を 0, 1, 2, . . . と変化させて，それぞれの場合に必要な硬貨の枚数の合計を

枚数(［ウ］)＋**枚数**(［エ］)

と計算し，一番小さな値を最小交換硬貨枚数とすればよいのです。

S：なるほど。それで，釣り銭 y はいくらまで調べればよいでしょうか？

T：面白い数学パズルですね。まあ，詳しくは今度考えるとして，今回は 100 円以下の商品なので y は 99 まで調べれば十分でしょう。

［イ］の解答群

⓪ **枚数**(51) ＋ **枚数**(5)　　① **枚数**(46) ＋ **枚数**(5)
② **枚数**(51) － **枚数**(5)　　③ **枚数**(46) － **枚数**(5)

［ウ］・［エ］の解答群

⓪ x　　① y　　② $x + y$　　③ $x - y$

［関数］
枚数(引数：金額)
　　・・・・・
戻り値：最小硬貨枚数
例）枚数(5) ⇒ 1
　　枚数(3) ⇒ 3
　　枚数(8) ⇒ 4
　　枚数(46)⇒ 6

46円の支払いの例
支払い：枚数(51)⇒2


釣り銭：枚数(5) ⇒1


計：交換硬貨枚数⇒3

価格 x 円, 釣り銭 y 円とすると，
支払いは，［　　］円

［反復］
※最小交換硬貨枚数を知るため，全ての場合を計算してみる。
釣り銭：0～99と変化
［支払い］　［釣り銭］
枚数(46) ＋ 枚数(0)
枚数(47) ＋ 枚数(1)
枚数(48) ＋ 枚数(2)
枚数(49) ＋ 枚数(3)
　　：
枚数(145)＋ 枚数(99)
全ての硬貨交換枚数のうち 一番小さな値
⇒ 最小交換硬貨枚数

【答】ア. 6　イ. 0：枚数(51)＋枚数(5)　ウ. 2：$x+y$　エ. 1：y

問２　次の文章の空欄［オ］～［コ］に入れるのに最も適当なものを，後の解答群のうちから一つずつ選べ。

Ｓ：まずは，関数「**枚数**(金額)」のプログラムを作るために，与えられた金額ちょうどになる最小の硬貨枚数を計算するプログラムを考えてみます。もう少しヒントが欲しいなぁ。

Ｔ：金額に対して，高額の硬貨から使うように考えて枚数と残金を計算していくとよいでしょう。また，金額に対して，ある額の硬貨が何枚まで使えて，残金がいくらになるかを計算するには，整数値の商を求める演算『÷』とその余りを求める演算『%』が使えるでしょう。例えば，46 円に対して 10 円玉が何枚まで使えるかは［オ］で，その際にいくら残るかは［カ］で求めることができますね。

Ｓ：なるほど！あとは自分でできそうです。

Ｓさんは，先生（Ｔ）との会話からヒントを得て，変数 **kingaku** に与えられた目標の金額（100 円以下）に対し，その金額ちょうどになる最小の硬貨枚数を計算するプログラムを考えてみた(図１)。ここでは例として目標の金額を 46 円としている。

配列 **Kouka** に硬貨の額を低い順に設定している。なお，配列の添字は **0** から始まるものとする。最低額の硬貨が 1 円玉なので **Kouka[0]**の値は **1** となる。

先生（Ｔ）のヒントに従い，高額の硬貨から何枚まで使えるかを計算する方針で，**(4)**～**(6)**行目のような繰返し文にした。この繰返しで，変数 **maisu** に支払いに使う硬貨の枚数の合計が計算され，変数 **nokori** に残りいくら支払えばよいか，という残金が計算される。

実行してみると［ア］が表示されたので，正しく計算できていることが分かる。いろいろな例で試してみたが，すべて正しく計算できていることを確認できた。

［算術演算子］
例）9÷3＝3　9%3＝0
　　8÷3＝2　8%3＝2
　　7÷3＝2　7%3＝1
　　6÷3＝2　6%3＝0
　　5÷3＝1　5%3＝2
　　4÷3＝1　4%3＝1
　　3÷3＝1　3%3＝0
　　2÷3＝0　2%3＝2
　　1÷3＝0　1%3＝1
　　0÷3＝0　0%3＝0

☞ Pythonでは
　整数値の商は『//』

［変数テーブル］
kingaku：目標金額
　　(100円以下)
Kouka：硬貨(配列)
　Kouka[0]⇒　1円玉
　Kouka[1]⇒　5円玉
　Kouka[2]⇒ 10円玉
　Kouka[3]⇒ 50円玉
　Kouka[4]⇒100円玉
maisu：使用硬貨合計
nokori：残りの額

［トレース］ 硬貨の枚数を考えるためのアルゴリズム（［増分処理］［反復］の考え方を含む）

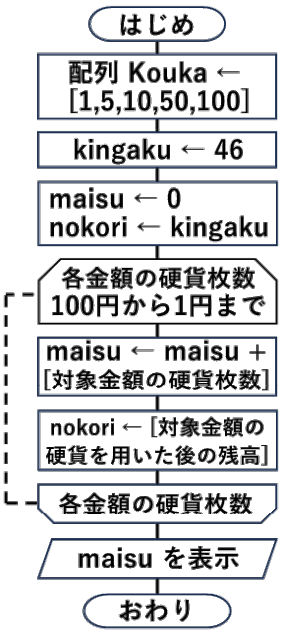
例）82円のとき

		maisu(初期値0)	nokori(初期値82)	具体的な意味
100円玉	82÷100＝0 82%100＝82	0 + 0 = 0	82	100円玉 0枚　硬貨合計 0枚 残額 82円
50円玉	82÷ 50＝1 82% 50＝32	0 + 1 = 1	32	50円玉 1枚　硬貨合計 1枚 残額 32円
10円玉	32÷ 10＝3 32% 10＝2	1 + 3 = 4	2	10円玉 3枚　硬貨合計 4枚 残額 2円
5円玉	2÷ 5＝0 2% 5＝2	4 + 0 = 4	2	5円玉 0枚　硬貨合計 4枚 残額 2円
1円玉	2÷ 1＝2 2% 1＝0	4 + 2 = 6	0	1円玉 2枚　硬貨合計 6枚 残額 0円

【答】オ.2：46÷10　カ.3：46%10

```
(1) Kouka = [1,5,10,50,100]
(2) kingaku = 46
(3) maisu = 0, nokori = kingaku
(4) i を [ キ ] ながら繰り返す:
(5)     maisu = [ ク ] + [ ケ ]
(6)     nokori = [ コ ]
(7) 表示する(maisu)
```

図１　目標の金額ちょうどになる最小の硬貨枚数を計算するプログラム



[オ]・[カ] の解答群

⓪ 46 ÷ 10 ＋ 1　　① 46 ％ 10 － 1
② 46 ÷ 10　　③ 46 ％ 10

[キ] の解答群

⓪ **5** から **1** まで **1** ずつ減らし　　① **4** から **0** まで **1** ずつ減らし
② **0** から **4** まで **1** ずつ増やし　　③ **1** から **5** まで **1** ずつ増やし

[ク] の解答群

⓪ **1**　　① **maisu**　　② **i**　　③ **nokori**

[ケ]・[コ] の解答群

⓪ **nokori ÷ Kouka[i]**　　① **nokori ％ Kouka[i]**
② **maisu ÷ Kouka[i]**　　③ **maisu ％ Kouka[i]**

初期値の設定

- kingaku は
  『例として46円としている』のだから
  最初は 46 に設定
- maisu は
  硬貨の枚数を順に加算していくので，
  最初は 0 に設定
  ※次第に大きくなる
- nokori は
  高い金額の硬貨順に必要な枚数を決定し，その都度の残高に当たる金額だから，
  最初は全金額
  ※反復する度に変化

DNCL 図1

```
Kouka = [1, 5, 10, 50, 100]
kingaku = 46
maisu = 0 , nokori = kingaku
i を 4 から 0 まで 1 ずつ減らしながら繰り返す:
| maisu = maisu + nokori ÷ Kouka[i]
└ nokori = nokori % Kouka[i]
表示する(maisu)
```

Python 図1　01

```
Kouka = [1, 5, 10, 50, 100]
kingaku = 46
maisu = 0 ; nokori = kingaku
for i in range(4, -1, -1):
  maisu = maisu + nokori // Kouka[i]
  nokori = nokori % Kouka[i]
print(maisu)
```

☞ ４行目を配列(リスト)で置き換え可能
→Python[02_参考]

【答】キ.1：4から0まで1ずつ減らし　ク.1：maisu　ケ.0：nokori÷Kouka[i]　コ.1：nokori%Kouka[i]

問３　次の文章を参考に，図２のプログラムの空欄 サ ～ タ に入れるのに最も適当なものを，後の解答群のうちから一つずつ選べ。ただし，空欄 ス ・ セ は解答の順序は問わない。

Ｔ：プログラム（図１）ができたようですね。それを使えば，関数「**枚数**(金額)」のプログラムができます。関数の引数として与えられる金額の値をプログラム（図１）の変数 **kingaku** に設定し，**(7)**行目の代わりに変数 **maisu** の値を関数の戻り値とすれば，関数「**枚数**(金額)」のプログラムとなります。では，その関数を使って最小交換硬貨枚数を計算するプログラムを作ってみましょう。ここでも，100 円以下の買い物として考えてみます。

【関数の説明】(再掲)

> **枚数**(金額)… 引数として「金額」が与えられ，ちょうどその金額となる硬貨の組合せの中で，枚数が最小となる硬貨枚数が戻り値となる関数。

Ｓさんは，図２のようなプログラムを作成した。変数 **kakaku** に与えられる商品の価格に対して，釣り銭を表す変数 **tsuri** を用意し，妥当な **tsuri** のすべての値に対して交換する硬貨の枚数を調べ，その最小値を求めるプログラムである。なお，ここでは例として商品の価格を 46 円としている。

このプログラムでは，先生（Ｔ）のアドバイスに従い，釣り銭無しの場合も含め，99 円までのすべての釣り銭に対し，その釣り銭になるように支払う場合に交換される硬貨の枚数を求め，その最小値を最小交換硬貨枚数として計算している。

最小値の計算では，これまでの払い方での最小枚数を変数 **min_maisu** に記憶しておき，それより少ない枚数の払い方が出るたびに更新している。**min_maisu** の初期値には，十分に大きな値として 100 を用いている。100 円以下の買い物では，使う硬貨の枚数は 100 枚を超えないからである。

変数テーブル
shiharai：支払いの額
kakaku：商品の価格
tsuri：釣り銭の額
※shiharai の額だけのお金を出し tsuri の額だけの返金を受け kakaku の額の商品を買う。
maisu：
　shiharai と tsuri でやりとりした硬貨の枚数の合計
min_masisu：
　それまでの払い方での最小硬貨交換枚数

Python 図2の関数 ― 03

```
def 枚数(kingaku):
  Kouka = [1,5,10,50,100]
  maisu = 0 ; nokori = kingaku
  for i in range(4,-1,-1):
    maisu = maisu + nokori // Kouka[i]
    nokori = nokori % Kouka[i]
  return maisu
```

トレース tsuri(0～99)とした色々な金額でのやりとり

shiharai	kakaku	tsuri	maisu	min_maisu
[START]				[初期値 100]
46	46	0	6	6
47	46	1	8	6
48	46	2	10	6
49	46	3	12	6
50	46	4	5	5
51	46	5	3	3
52	46	6	5	3
:	:	:	:	:
145	46	99	10	?

```
(1) kakaku = 46
(2) min_maisu = 100
(3) [サ] を [シ] から 99 まで 1 ずつ増やしながら繰り返す：
(4)   shiharai = kakaku + tsuri
(5)   maisu = [ス] + [セ]
(6)   もし [ソ] < min_maisu ならば：
(7)     [タ] = ソ
(8) 表示する(min_maisu)
```

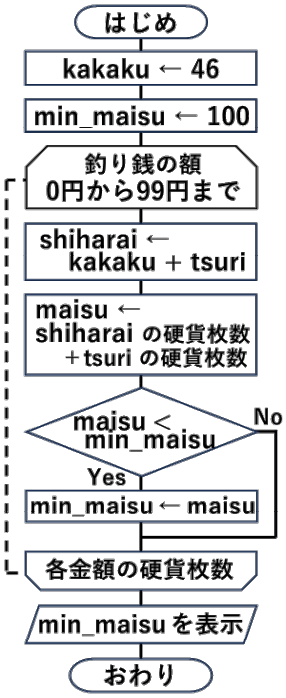


図２　最小交換硬貨枚数を求めるプログラム

このプログラムを実行してみたところ 3 が表示された。46 円を支払うときの最小交換硬貨枚数は，支払いで 50 円玉が 1 枚，1 円玉が 1 枚，釣り銭で 5 円玉が 1 枚の計 3 枚なので，正しく計算できていることが分かる。同様に，**kakaku** の値をいろいろと変えて実行してみたところ，すべて正しく計算できていることを確認できた。

[サ], [ソ]・[タ] の解答群

⓪ **maisu**　① **min_maisu**　② **shiharai**　③ **tsuri**

[シ] の解答群

⓪ 0　① 1　② 99　③ 100

[ス]・[セ] の解答群

⓪ 枚数(**shiharai**)　① 枚数(**kakaku**)　② 枚数(**tsuri**)
③ **shiharai**　④ **kakaku**　⑤ **tsuri**

DNCL 図2

```
kakaku = 46
min_maisu = 100
i を 0 から 99 まで 1 ずつ増やしながら繰り返す
  | shiharai = kakaku + tsuri
  | maisu = 枚数(shiharai) + 枚数(tsuri)
  | もし miasu < min_maisu
  |   min_maisu = maisu
表示する(min_maisu)
```

Python 図2　03

```
kakaku = 46
min_maisu = 100
for tsuri in range(100):
  shiharai = kakaku + tsuri
  maisu = 枚数(shiharai) + 枚数(tsuri)
  if maisu < min_maisu:
    min_maisu = maisu
print(min_maisu)
```

【答】サ.3：tsuri　シ.0：0　ス.0：枚数(shiharai)　セ.2：枚数(tsuri)　ソ.0：maisu　タ.1：min_maisu

②令和３年 文部科学省 サンプル問題

第２問 次の文章を読み，後の問い(問１～３)に答えよ。

Mさんは，18 歳になって選挙権が得られたのを機に，比例代表選挙の当選者を決定する仕組みに興味を持った。そこで各政党に配分する議席数（当選者数）を決める方法を，友人のKさんとプログラムを用いて検討してみることにした。

問１ 次の文章の空欄 **ア** ～ **ウ** に入れる最も適当なものを，後の解答群のうちから一つずつ選べ。同じものを繰り返し選んでもよい。

Mさん：表１に，最近行われた選挙結果のうち，ある地域のブロックについて，各政党の得票数を書いてみたよ。

表１　各政党の得票数

	A党	B党	C党	D党
得票数	1200	660	1440	180

Kさん：今回の議席数は６人だったね。得票の総数を議席数で割ると 580 人なので，これを基準得票数と呼ぶのがいいかな。平均して１議席が何票分の重みがあるかを表す数ということで。そうすると，各政党の得票数が何議席分に相当するかは，各政党の得票数をこの基準得票数で割れば求められるね。

Mさん：その考え方に沿って政党ごとの当選者数を計算するプログラムを書いてみよう。まず，プログラムの中で扱うデータを図１と図２にまとめてみたよ。配列 Tomei には各政党の党名を，配列 Tokuhyo には各政党の得票数を格納することにしよう。政党の数は４つなので，各配列の添字は 0 から 3 だね。

i	0	1	2	3
Tomei	A党	B党	C党	D党

図１　各政党名が格納されている配列

i	0	1	2	3
Tokuhyo	1200	660	1440	180

図２　得票数が格納されている配列

Mさん：では，これらのデータを使って，各政党の当選者数を求める図３のプログラムを書いてみよう。実行したら図４の結果が表示されたよ。

変数テーブル

Tomei：党名(配列)
  Tomei[0]="A党"
  Tomei[1]="B党"
  Tomei[2]="C党"
  Tomei[3]="D党"

Tokuhyo：
各政党の得票数(配列)
  Tokuhyo[0]=1200
  Tokuhyo[1]= 660
  Tokuhyo[2]=1440
  Tokuhyo[3]= 180

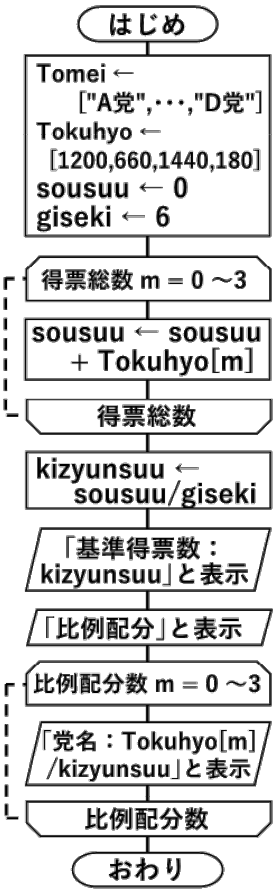
sousuu：総得票数

giseki：総議席数

kizyunsuu：
１議席の重み(得票数)

```
(01) Tomei = ["A党","B党","C党","D党"]
(02) Tokuhyo = [1200,660,1440,180]
(03) sousuu = 0
(04) giseki = 6
(05) mを0から [ア] まで1ずつ増やしながら繰り返す:
(06)     sousuu = sousuu + Tokuhyo[m]
(07) kizyunsuu = sousuu / giseki
(08) 表示する("基準得票数:",kizyunsuu )
(09) 表示する("比例配分")
(10) mを0から [ア] まで1ずつ増やしながら繰り返す:
(11)     表示する(Tomei[m],":",[イ] / [ウ] )
```

図３　得票に比例した各政党の当選者数を求めるプログラム



Kさん：得票数に比例して配分すると小数点のある人数になってしまうね。小数点以下の数はどう考えようか。例えば，A党は2.068966だから２人が当選するのかな。

Mさん：なるほど。切り捨てで計算すると，A党は２人，B党は１人，C党は２人，D党は０人になるね。あれ？ 当選者数の合計は５人で，６人に足りないよ。

```
基準得票数：580
比例配分
A党：2.068966
B党：1.137931
C党：2.482759
D党：0.310345
```

図４　各政党の当選者数の表示

Kさん：切り捨ての代わりに四捨五入したらどうだろう。

Mさん：そうだね。ただ，この場合はどの政党も小数点以下が0.5未満だから，切り捨てた場合と変わらないな。だからといって小数点以下を切り上げると，当選者数が合計で９人になるから３人も多くなってしまう。

Kさん：このままでは上手くいかないなぁ。先生に聞いてみよう。

[ア]～[ウ]の解答群

⓪ 0　① 1　② 2　③ 3　④ 4　⑤ 5　⑥ 6　⑦ Tomei[m]
⑧ Tokuhyo[m]　⑨ sousuu　ⓐ giseki　ⓑ kizyunsuu

[トレース] 得票総数を求めるための反復

	Tokuhyo	sousuu
		初期値 0
A党	1200	1200
B党	660	1860
C党	1440	3300
D党	180	3480

↑ 反復を抜けた後の値

DNCL 図3

```
Tomei = ["A党","B党","C党","D党"]
Tokuhyo = [1200,660,1440,180]
sousuu = 0
giseki = 6
mを0から3まで1ずつ増やしながら繰り返す:
    sousuu = sousuu + Tokuhyo[m]
kizyunsuu = sousuu / giseki
表示する("基準得票数:",kizyunsuu)
表示する("比例配分")
mを0から3まで1ずつ増やしながら繰り返す:
    表示する(Tomei[m],":",Tokuhyo[m]/kizyunsuu)
```

Python 図3　04

```
Tomei = ["A党","B党","C党","D党"]
Tokuhyo = [1200,660,1440,180]
sousuu = 0
giseki = 6
for m in range(4):
    sousuu = sousuu + Tokuhyo[m]
kizyunsuu = sousuu / giseki
print("基準得票数:",kizyunsuu)
print("比例配分")
for m in range(4):
    print(Tomei[m],":",Tokuhyo[m]/kizyunsuu)
```

【答】ア.3：3　イ.8：Tokuhyo[m]　ウ.b：kizyunsuu

問２　次の文章の空欄 エ ～ ス に入れる最も適当なものを，後の解答群のうちから一つずつ選べ。同じものを繰り返し選んでもよい。

Mさん：先生，比例代表選挙では各政党の当選者数はどうやって決まるのですか？ 当選者数が整数なので，割合だけだと上手くいかなかったのです。

先　生：様々な方法があるけど，日本では各政党の得票数を１，２，３，…と整数で割った商の大きい順に定められた議席を配分していく方法を採用しているよ。この例だと表２のように，❶から❻の順に議席が各政党に割り当てられるんだ。Ｃ党が❶の議席を取っているけど，このとき，何の数値を比較したか分かるかな。

表２　各政党の得票数と整数で割った商

	Ａ党	Ｂ党	Ｃ党	Ｄ党
得票数	1200	660	1440	180
１で割った商	❷1200	❹ 660	❶1440	180
２で割った商	❺ 600	330	❸ 720	90
３で割った商	400	220	❻ 480	60
４で割った商	300	165	360	45

Mさん：１で割った商です。Ａ党から順に 1200，660，1440，180 ですね。

先　生：そうだね。ではＡ党が❷の議席を取るとき，何の数値を比較したのだろうか。

Mさん：Ｃ党は１議席目を取ったので，1440 を２で割った商である 720 を比較します。Ａ党から順に 1200，660，720，180 ですね。この中で数値が大きいＡ党が議席を取ります。なるほど，妥当な方法ですね。

Kさん：この考え方で手順を考えてみようよ。

先　生：まずは候補者が十分足りるという条件で手順を考えてみるのがいいですよ。

Kさん：各政党に割り当てる議席を決めるために，比較する数値を格納する配列 Hikaku がいるね。

Mさん：各政党に配分する議席数（当選者数）を格納する配列 Tosen も必要だね。最初は議席の配分が行われていないから，初期値は全部 0 にしておくね。

i	0	1	2	3
Hikaku				

図５　整数で割った値を格納する配列

i	0	1	2	3
Tosen	0	0	0	0

図６　当選者数を格納する配列

変数テーブル

Hikaku：（配列）
　配分を比較する数値
Tosen：（配列）
　当選者数
　※初期値は０に設定

Kさん：「２で割った商」の「２」のように，各政党の得票数を割るときに使う数字はどうすればいいかな。

Mさん：その政党の当選者数＋１でいいよね。配列 Tosen が使えるね。そうだ，変化したところだけ計算し直せばいいんじゃない？ 議席を配分する手順を書いてみよう。

手順1　配列 Tokuhyo の各要素の値を配列 Hikaku の初期値として格納する。
手順2　配列 Hikaku の要素の中で最大の値を調べ，その添字 maxi に対応する配列 Tosen[maxi] に１を加える。
手順3　Tokuhyo[maxi] を Tosen[maxi]＋1 で割った商を Hikaku[maxi] に格納する。
手順4　手順2と手順3を当選者数の合計が議席数の6になるまで繰り返す。
手順5　各政党の党名（配列 Tomei）とその当選者数（配列 Tosen）を順に表示する。

図７　手順を書き出した文章

Kさん：この図７の手順が正しいか確認するために，配列 Hikaku と配列 Tosen の中がどう変化していくか確認してみよう。図８のようになるね。

配列 Hikaku の変化

i	0	1	2	3
手順1終了時	1200	660	1440	180
1回目の手順3終了時	1200	660	720	180
2回目の手順3終了時	600	660	エ	180
3回目の手順3終了時	600	660	オ	180
4回目の手順3終了時	600	330	カ	180
5回目の手順3終了時	400	330	キ	180
6回目の手順3終了時	400	330	ク	180

配列 Tosen の変化

i	0	1	2	3
手順1終了時	0	0	0	0
1回目の手順3終了時	0	0	1	0
2回目の手順3終了時	1	0	ケ	0
3回目の手順3終了時	1	0	コ	0
4回目の手順3終了時	1	1	サ	0
5回目の手順3終了時	2	1	シ	0
6回目の手順3終了時	2	1	ス	0

図８　配列 Hikaku と配列 Tosen の変化

Mさん：先生に教えてもらった結果と同じように，議席数が６になるまで議席を配分できたね。この手順でプログラムを考えてみよう。

エ ～ ス の解答群

⓪ 0　① 1　② 2　③ 3　④ 4　⑤ 180
⑥ 288　⑦ 360　⑧ 400　⑨ 480　ⓐ 600　ⓑ 720

変数テーブル
tosenkei：
　決定した当選者の計

トレース　図８　配列Hikaku と 配列 Tosen の変化（番号順に考えると良い）

配列 Hikaku の変化

i	0	1	2	3
手順1終了時	1200	660	1440	180
1回目の手順3終了時	1200	660	② 720	180
2回目の手順3終了時	⑤ 600	660	⑥	180
3回目の手順3終了時	600	660	⑧	180
4回目の手順3終了時	600	⑪ 330	⑫	180
5回目の手順3終了時	⑮ 400	330	⑯	180
6回目の手順3終了時	400	330	⑱	180

配列 Tosen の変化

i	0	1	2	3
手順1終了時	0	0	0	0
1回目の手順3終了時	0	0	① 1	0
2回目の手順3終了時	③ 1	0	④	0
3回目の手順3終了時	1	0	⑦	0
4回目の手順3終了時	1	⑨ 1	⑩	0
5回目の手順3終了時	⑬ 2	1	⑭	0
6回目の手順3終了時	2	1	⑰	0

⑥ 720　④ 1
⑧ 480　⑦ 2
⑫ 480　⑩ 2
⑯ 480　⑭ 2
⑱ 360　⑰ 3

【答】エ. b：720　オ. 9：480　カ. 9：480　キ. 9：480　ク. 7：360　ケ. 1：1　コ. 2：2　サ. 2：2　シ. 2：2　ス. 3：3

問３　次の文章の空欄 セ ～ テ に入れる最も適当なものを，後の解答群のうちから一つずつ選べ。

Mさん：図９のプログラムを作ってみたよ。商を整数で求めるところは小数点以下を切り捨てる「**切り捨て**」という関数を使ったよ。

Kさん：実行したら図 10 のように正しく政党名と当選者数が得られたね。

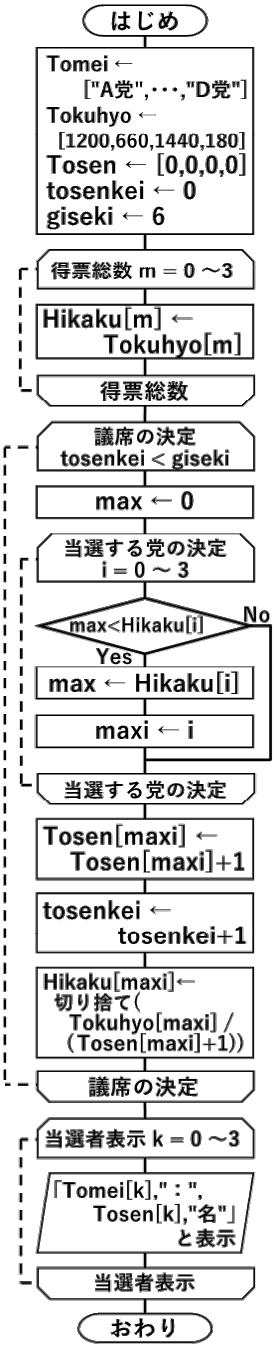
```
(01) Tomei = ["A 党","B 党","C 党","D 党"]
(02) Tokuhyo = [1200,660,1440,180]
(03) Tosen = [0,0,0,0]
(04) tosenkei = 0
(05) giseki = 6
(06) m を 0 から [ ア ] まで 1 ずつ増やしながら繰り返す:
(07)     Hikaku[m] = Tokuhyo[m]
(08) [ セ ] < giseki の間繰り返す:
(09)     max = 0
(10)     i を 0 から [ ア ] まで 1 ずつ増やしながら繰り返す:
(11)         もし max < Hikaku[i]ならば:
(12)             [ ソ ]
(13)             maxi = i
(14)     Tosen[maxi] = Tosen[maxi] + 1
(15)     tosenkei = tosenkei + 1
(16)     Hikaku[maxi] = 切り捨て( [ タ ] / [ チ ] )
(17) k を 0 から [ ア ] まで 1 ずつ増やしながら繰り返す:
(18)     表示する(Tomei[k],"：",Tosen[k],"名")
```

図９　各政党の当選者数を求めるプログラム

先　生：できたようだね。各政党の当選者数は求められたけど，政党によっては候補者が足りない場合もあるから，その場合にも対応してみよう。図 11 のように各政党の候補者数を格納する配列 Koho を追加してみたらどうだろう。例えば，C党の候補者が足りなくなるように設定してみよう。

```
A党:2 名
B党:1 名
C党:3 名
D党:0 名
```

図 10　各政党の当選者数の表示



DNCL 図9

```
Tomei = [″A党″,″B党″,″C党″,″D党″]
Tokuhyo = [1200,660,1440,180]
Tosen = [0,0,0,0]

tosenkei = 0
giseki = 6
mを0から3まで1ずつ増やしながら繰り返す:
  └ Hikaku[m] = Tokuhyo[m]
tosenkei < giseki の間繰り返す:
  │ max = 0
  │ iを0から3まで1ずつ増やしながら繰り返す:
  │   │ もし max < Hikaku[i] ならば:
  │   │   │ max = Hikaku[i]
  │   └   └ maxi = i
  │ Tosen[maxi] = Tosen[maxi] + 1
  │ tosenkei = tosenkei +1
  └ Hikaku[maxi]=切り捨て(Tokuhyo[maxi]/(Tosen[maxi]+1))
mを0から3まで1ずつ増やしながら繰り返す:
  └ 表示する(Tomei[k],″″,Tosen[k],″名″)
```

Python 図9　05

```
def 切り捨て(x):                    #関数の定義
  return x//1
Tomei = [″A党″,″B党″,″C党″,″D党″]
Tokuhyo = [1200,660,1440,180]
Tosen = [0,0,0,0]
Hikaku = [0,0,0,0]                  #リスト(配列)の生成
tosenkei = 0
giseki = 6
for m in range(4):
  Hikaku[m] = Tokuhyo[m]
while tosenkei < giseki:
  max = 0
  for i in range(4):
    if max < Hikaku[i]:
      max = Hikaku[i]
      maxi = i
  Tosen[maxi] = Tosen[maxi] + 1
  tosenkei = tosenkei +1
  Hikaku[maxi]=切り捨て(Tokuhyo[maxi]/(Tosen[maxi]+1))
for k in range(4):
  print(Tomei[k],″″,Tosen[k],″名″)
```

☞ 関数「切り捨て( )」を用いず，DNCL:Tokuhyo[maxi]÷(Tosen[maxi]+1),Python:Tokuhyo[maxi]//(Tosen[maxi]+1)でもよい。

【答】セ.2：tosenkei　ソ.2：max=Hikaku[i]　タ.3：Tokuhyo[maxi]　チ.8：(Tosen[maxi]+1)

i	0	1	2	3
Koho	5	4	2	3

図 11　候補者数を格納する配列

Mさん：候補者が足りなくなったらどういう処理をすれば良いのですか？

先　生：比較した得票で次に大きい得票数の政党が繰り上がって議席を取るんだよ。

Mさん：なるほど。では，図９の(11)行目の条件文を次のように修正すればいいですね。当選していない候補者はどこかの政党には必ずいるという前提だけど。

```
(11)  |  |  もし max < Hikaku[i]  ツ  テ  ならば:
```

Kさん：先生，候補者が不足するほかに，考えるべきことはありますか？

先　生:例えば,配列Hikakuの値が同じになった政党の数が残りの議席の数より多い場合，このプログラムでは添字の小さい政党に議席が割り当てられてしまうので不公平だね。実際には，この場合はくじ引きで議席を割り当てるようだよ。

セ，**タ**・**チ**の解答群

⓪ max	① maxi	② tosenkei
③ Tokuhyo[maxi]	④ Tokuhyo[maxi] + 1	⑤ Tokuhyo[max]
⑥ Tosen[maxi]	⑦ Tosen[maxi + 1]	⑧ (Tosen[maxi] + 1)

ソの解答群

⓪ max = max + 1	① max = Tokuhyo[i]	② max = Hikaku[i]
③ Hikaku[i] = max	④ Tokuhyo[i] = max	⑤ Tokuhyo[i] = Hikaku[i]

ツの解答群

⓪ and	① or	② not

テの解答群

⓪ Koho[i] >= Tosen[i] + 1	① Koho[i] < Tosen[i] + 1
② Koho[i] >= Tosen[i]	③ Koho[i] < Tosen[i]

候補者が十分にいる場合

	候補者数	当選者数
A党	∞	2
B党	∞	1
C党	∞	3
D党	∞	0

候補者数が限られる場合

	候補者数	当選者数
A党	5	3
B党	4	1
C党	2	2
D党	3	0

【発展】
左のような場合は，
どのようなプログラムにすればよいだろうか。

※下線部を追加･修正する

DNCL

```
      :
Tomei = ["A党","B党","C党","D党"]
Tokuhyo = [1200,660,1440,180]

Tosen = [0,0,0,0]
      :
      :
  | iを0から3まで1ずつ増やしながら繰り返す:
  |   | もしmax<Hikaku[i] and Koho[i]>=Tosen[i]+1:ならば
  |   |   | max = Hikaku[i]
  |   |   | maxi = i
      :
```

Python 06

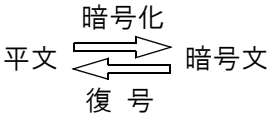
```
      :
Tomei = ["A党","B党","C党","D党"]
Tokuhyo = [1200,660,1440,180]
Koho = [5,4,2,3]
Tosen = [0,0,0,0]
      :
      :
  for i in range(4):
    if max<Hikaku[i] and Koho[i]>=Tosen[i]+1:
      max = Hikaku[i]
      maxi = i
      :
```

【答】ツ.0：and　テ.0：Koho[i]>=Tosen[i]+1

③令和２年 情報処理学科 試作検討問題

第５問　次の文章を読み，後の問い(**問 1～3**)に答えよ。

シーザー暗号に代表される古典的な暗号化の方法であるシフト暗号はアルファベットの文字を決まった文字数分シフトさせて（ずらして）置き換える極めて単純な暗号手段である。T さんと M さんは授業で先生が出した課題であるシフト暗号で暗号化した暗号文をいかに解読するかを考えることにした。



問１　次の会話文を読み，空欄 アイ ～ キク に当てはまる数字をマークせよ。

課題　英文をシフト暗号で暗号化した以下の暗号文を解読しなさい。ただし，英文は全て小文字でアルファベット以外のスペースや数字，「'」「,」「.」「?」などは変換されていません。

（省略）‥‥nonsmkdo k zybdsyx yp drkd psovn, kc k psxkv bocdsxq zvkmo pyb dryco gry robo qkfo drosb vsfoc drkd dro xkdsyx wsqrd vsfo. sd sc kvdyqodrob psddsxq kxn zbyzob drkd go cryevn ny drsc. led, sx k vkbqob coxco, go mkx xyd nonsmkdo - go mkx xyd myxcombkdo - go mkx xyd rkvvyg - drsc qbyexn. dro lbkfo wox, vsfsxq kxn nokn, gry cdbeqqvon robo, rkfo myxcombkdon sd, pkb klyfo yeb zyyb zygob dy knn yb nodbkmd. dro gybvn gsvv vsddvo xydo, xyb vyxq bowowlob grkd go cki robo, led sd mkx xofob pybqod grkd droi nsn robo. sd ‥‥（省略）

図１　先生が出した課題

M さん:シフト暗号って，例えばアルファベットを５文字右にシフトした場合，文字「a」は文字「f」に，文字「x」はまず２文字シフトして右端に達した後一番左端に戻り３文字シフトした文字「c」に置き換わるやつだよね。暗号化された文字列の復号は，その逆，つまり左に５文字シフトすればできるね。

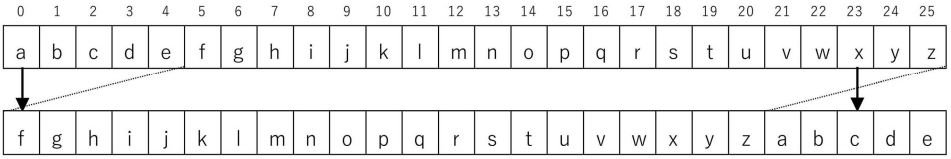


図２　５文字右シフトした場合の考え方

T さん:復号は必ずしも反対にシフトする必要はないよね。例えば 9 文字右にシフトされていた場合，復号するには 9 文字左にシフトでも良いけど，右に［アイ］文字シフトすることもできるね。図 2 のようにアルファベットに 0～25 の番号を割り当てて考えてみると，暗号化して x 番目の文字になった時，復号は x+［アイ］の値が［ウエ］以下であれば x+［アイ］番の文字に置き換わるけど，［ウエ］より大きい場合は，x+［アイ］−［オカ］番の文字に置き換えれば復号できるよね。

M さん:暗号化で文字を何文字シフトしているか分かれば，この復号法で解読できるよね。どうやったら分かるかな。

T さん:すべての可能性，つまりシフトしない時を除いた［キク］通りをプログラムで試せばいいんじゃない？

M さん:この場合だと［キク］通りで済むけども，大文字があったり，日本語のように文字種の数が多い言語ではとても効率が悪い方法だよ。英文であれば，単語に含まれる「a」とか「e」が多い気がするし，逆に「z」が含まれる単語は少ししか思いつかない。アルファベットの出現頻度を調べればある程度推測できるんじゃないかな。インターネットで調べてみようよ。

M さん:どうやら一般的な英文のアルファベットの出現頻度には図 3 のような傾向があるみたいだよ。

T さん:文字によって出現頻度に特徴があるね。暗号化された英文のアルファベットの出現頻度を調べれば，何文字シフトされているか推測することができそうだね。一つ一つ数え上げるのは大変だから数え上げるプログラムを考えてみるよ。

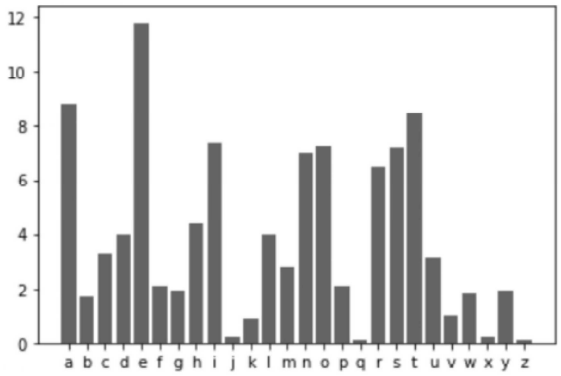
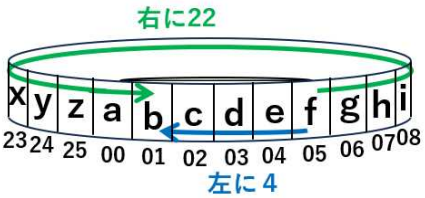


図3 出現頻度のグラフ （縦軸%）



この関係は26文字で一周し元に戻るのだから，左への移動量と右への移動量は『26を法とする合同』の関係となっている。

実際，文字識別番号 x の文字を左に n 文字移動させる場合

左に 1 文字移動 ⇒ 右に 25 文字移動 ： x − 1 ＝ x ＋ 2 5 − 2 6

左に 2 文字移動 ⇒ 右に 24 文字移動 ： x − 2 ＝ x ＋ 2 4 − 2 6

左に 3 文字移動 ⇒ 右に 23 文字移動 ： x − 3 ＝ x ＋ 2 3 − 2 6

左に 4 文字移動 ⇒ 右に 22 文字移動 ： x − 4 ＝ x ＋ 2 2 − 2 6

：

左に n 文字移動 ⇒ 右に［　］文字移動 ； x − n ＝ x ＋［　］−［　］

【答】ア. 1：1　イ. 7：7　ウ. 2：2　エ. 5：5　オ. 2：2　カ. 6：6　キ. 2：2　ク. 5：5

問 2　次の会話文を読み，空欄 ケ ・ コ に当てはまる内容を，後の解答群のうちから一つずつ選べ。また，空欄 サシ に当てはまる数字をマークせよ。

T さん:暗号化された英文のアルファベットの出現頻度を数え上げるプログラムを図 5 のように考えてみたよ。このプログラムでは，配列変数 Angoubun に暗号文を入れて，一文字ずつアルファベットの出現頻度を数え上げて，その結果を配列変数 Hindo に入れているんだ。Hindo[0]が「a」，Hindo[25]が「z」に対応しているよ。

x	0	1	2	3	4	5	6	7	8	……	20	21	22	23	24	25
Hindo[x]	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0

図4　アルファベットの出現頻度を数え上げる配列

```
(01)Angoubun = ["p","y","e","b",…(省略)…"k","b","d","r","."]
(02)配列 Hindo のすべての要素に 0 を代入する
(03)i を 0 から 要素数(Angoubun)-1 まで 1 ずつ増やしながら:
(04)  bangou = 差分( ケ )
(05)  もし bangou != −1 ならば:
(06)    コ = コ + 1
(07)表示する(Hindo)
```

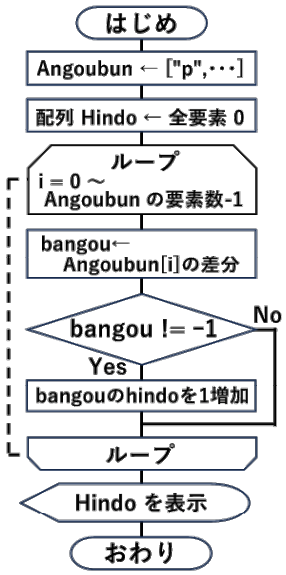
図5　出現頻度を求めるプログラム

【関数の説明】

要素数(値)…配列の要素数を返す。
例：Data=["M","i","s","s","i","s","s","i","p","p","i"]の時
要素数(Data)は 11 を返す

差分(値)…アルファベットの「a」との位置の差分を返す
値がアルファベット以外の文字であれば－1 を返す
例：差分("e")は 4 を，差分("x")は 23 を返す
差分("5")や差分(",")は－1 を返す

変数テーブル
Angoubun：暗号文（配列）
Hindo　：出現頻度（配列）
bangou：文字識別番号



関数「要素数( )」の例

Python 07
```
def 要素数(Data):
 n = 0
 for i in Data:
   n = n + 1
 return n
```

関数「差分( )」の例

Python 08
```
def 差分(alp):
 n=ord(alp)-ord("a")
 if n<0 or n>25:
   n=-1
 return n
```

DNCL 図5
```
Angoubun = ["p","y","e",・・・(省略)・・・,"d","r","."]
配列Hindo のすべての要素に0を代入する
i を 0 から 要素数(Angoubun)-1 まで 1 ずつ増やしながら:
  | bangou = 差分(Angoubun[i])
  | もし bangou != -1 ならば:
  |   Hindo(bangou) = Hindo(bangou)
表示する(Hindo)
```

Python 図5　09(一部)
```
Angoubun = ["p","y",・・・(省略)・・・,"r","."]
Hindo = [0,0,0,・・・,0]
for i in range(要素数(Angoubun)):
  bangou = 差分(Angoubun[i])
  if bangou !=-1:
    Hindo[bangou]=Hindo[bangou]+1
print(Hindo)
```

☞ Python 図5 に以下のような変更を加えたものが Pythonプログラム 09 である。
① Pythonプログラム 07関数「要素数( )」，08関数「差分( )」 を追加する。
② リスト(配列)Angoubunについては，内容を変更した文字データを組み込み関数 list( ) によってリストに変換し設定する。なお，文字データが長文となるので，途中改行しないようトリプルクォーテーション"""で囲む。
③ リスト(配列) Hindo の生成は，Hindo=[0]*26 として繰り返してリストを生成するリストの複製手法を用いる。
④ 結果をグラフ化するために，以下のプログラムを追加する。

```
import matplotlib.pyplot as plt
x = ["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"]
plt.bar(x, Hindo)
plt.show()
```

【答】ケ.0：Angoubun[i]　　コ.4：Hindo(bangou)

M さん：これでアルファベットの出現頻度が調べられるね。それで結果はどうなったの？

T さん：このプログラムで得られた配列 Hindo をグラフ化してみたよ（図６）。

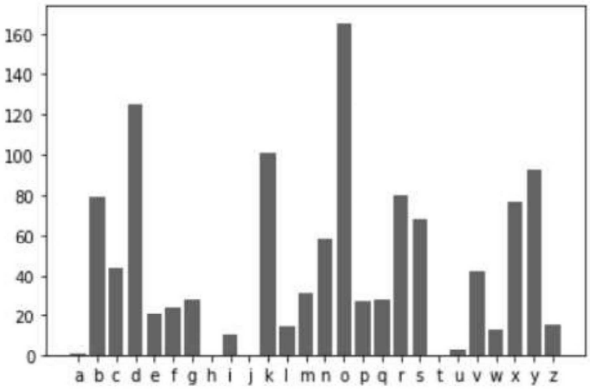


図６　アルファベットと配列 Hindo のグラフ表示

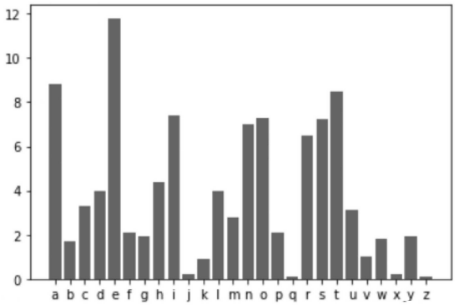
M さん：このアルファベットの出現頻度を見ると，「o」「d」「k」「y」が多いね。逆に出現頻度がない「a」「h」「j」「t」も手掛かりになるね。図３と照らし合わせると，この暗号化された文字列は右に［サシ］文字シフトしていると考えられるね。

T さん：うん，でもそれが正しいか，実際にプログラムを作って復号してみようよ。

［ケ］・［コ］の解答群

⓪ Angoubun[i]　① Angoubun[i-1]　② Angoubun[bangou]
③ Angoubun[bangou-1]　④ Hindo[bangou]　⑤ Hindo[bangou-1]
⑥ Hindo[i]　⑦ Hindo[i-1]

図３ 出現頻度のグラフ （縦軸％）



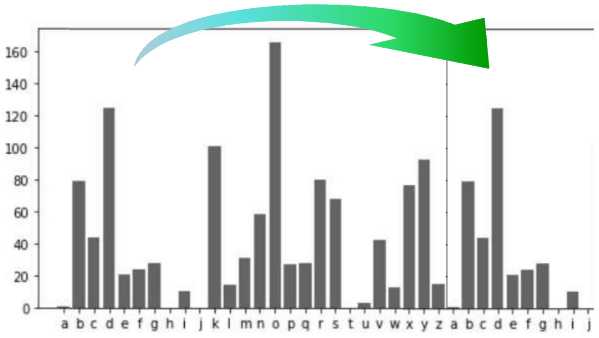


図６　アルファベットと配列 Hindo のグラフ表示

図6(下の図)　図3(上の図)
暗号文のHindo　一般的な出現頻度

o ----------> e
d ----------> t
k ----------> a
y ----------> o

a ----------> q
h ----------> x
j ----------> z
t ----------> j

暗号文 Hindo　abcdefghijklmnopqrstuvwxyzabcdefghijk

一般的（平文）　abcdefghijklmnopqrstuvwxyz

【答】サ. 1：1　シ. 0：0

問 3　次の会話文の空欄［ス］～［チ］に当てはまる内容を，後の解答群のうちから一つずつ選べ。

Tさん:暗号文を一文字ずつ復号して表示するプログラムができたよ（図7）。

Mさん:なるほど，復号も右にシフトで考えているんだね。実行してみたら読み取れる英文になったの？

```
(01)Angoubun = ["p","y","e","b",…(省略)…"k","b","d","r","."]
(02)配列変数 Hirabun を初期化する
(03)hukugousuu = 26 - [サシ]
(04)i を 0 から 要素数(Angoubun)-1 まで 1 ずつ増やしながら:
(05)  bangou = 差分([ク])
(06)  もし bangou != -1 ならば:
(07)    もし [ス] <= 25 ならば:
(08)      Hirabun[i] = 文字([ス])
(09)    そうでなければ:
(10)      Hirabun[i] = 文字([セ])
(11)  そうでなければ:
(12)    Hirabun[i] = [ソ]
(13)表示する(Hirabun)
```

※

図7　暗号文を復号するプログラム

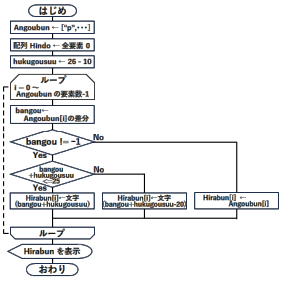
【関数の説明】

文字(値)…番号の値に対するアルファベットの文字を返す。
　　値が 0 以上 25 以下でなければ「アルファベットでない」を返す
例：**文字**(4)は「e」を，**文字**(23)は「x」を返す
　　文字(-1)や**文字**(27)は「アルファベットでない」を返す

Tさん:うん，復号したらこんな英文が表示されたよ。正しい英単語に変換されているみたいだから推測は当たっていたね。

four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.now we are engaged in a great civil war, testing whether that nation, or any nation ·····（省略） ····· last full measure of devotion - that we here highly resolve that these dead shall not have died in vain - that this nation, under god, shall have a new birth of freedom - and that government of the people, by the people, for the people, shall not perish from the earth.

変数テーブル
Hukugousuu：シフト数
Hirabun　：平文
（配列）



☞ 拡大図は次頁下段

【比較演算子】
DNCLとPythonは同じ

等しい	＝＝
等しくない	！＝
より大	＞
より小	＜
以　上	＞＝
以　下	＜＝

関数「文字( )」の例

Python　10
```
def 文字(num):
 alp="アルファベットでない"
 if num>=0 and num<=25:
   code=num+ord("a")
   alp=(chr(code))
 return alp
```

DNCL　図7
```
Angoubun = ["p","y","e",…(省略)…,"d","r","."]
配列変数 Hirabun を初期化する
hukugousuu = 26 - 10
i を 0 から 要素数(Angoubun)-1 まで 1 ずつ増やしながら:
  bangou = 差分(Angoubun[i])
  もし bangou != -1 ならば:
    もし bango + hukugousuu <= 25  ならば:
      Hirabun[i]=文字(bango + hukugousuu)
    そうでなければ:
      Hirabun[i]=文字(bango + hukugousuu - 26)
  そうでなければ:
    Hirabun[i] = Angoubun[i]
表示する(Hirabun)
```

Python　図7　11(一部)
```
Angoubun = ["p","y","e",…(省略)…,"r",".","."]
Hirabun = [""]*要素数(Angoubun)
hukugousuu = 26 - 10
for i in range(要素数(Angoubun)):
  bangou = 差分(Angoubun[i])
  if bangou != -1:
    if bangou + hukugousuu <= 25:
      Hirabun[i] = 文字(bangou + hukugousuu)
    else:
      Hirabun[i] = 文字(bangou + hukugousuu -26)
  else:
    Hirabun[i] = Angoubun[i]
print(Hirabun)
```

☞ Python 図7 に以下のような変更を加えたものが Pythonプログラム 09 である。
① Pythonプログラム 07関数「要素数( )」，08関数「差分( )」，10関数「文字( )」を追加する。
② リスト(配列)Angoubunについては，内容を変更した文字データを組み込み関数 list( ) によってリストに変換し設定する。なお，文字データが長文となるので，途中改行しないようトリプルクォーテーション"""で囲む。
③ リスト(配列) Hirabun の生成は，Hirabun=[""]*要素数(Angoubun) として繰り返しのリスト生成手法を用いる。
④ 出力については，文字を連結して表示させるためのプログラムコードを追加する。

【答】ス. 0：bangou+hukugousuu　セ. 3：bangou + hukugousuu - 26　ソ. 6：Angoubun[i]

Mさん:これって有名なリンカーンのゲティスバーグ演説じゃない。ほら最後のところ有名なフレーズだよね。

Tさん:先生,課題ができました。元の英文はリンカーンのゲティスバーグ演説ですね。プログラムで文字の出現頻度を調べて,シフトされた文字数を推測しました。復号はこのプログラムで変換してみました。

先生:よくできたね,素晴らしい!このプログラムはもっと簡単にできるね。この(07)～(10)の※部分は工夫すれば1行にまとめられるよ。ヒントは余りを求める算術演算子%を使うんだ。

Tさん:えっ, 1行ですか?・・・・・分かった!

Hirabun[i] = 文字(タ % チ)

とすればもっと簡潔にできたんだ。

先生:素晴らしい!

ス～**ソ**の解答群

⓪ bangou+hukugousuu　① bangou
② hukugousuu　③ bangou+hukugousuu-26
④ bangou+hukugousuu-25　⑤ hukugousuu-26
⑥ Angoubun[i]　⑦ Hirabun[i]
⑧ Angoubun[i+hukugousuu]

タの解答群

⓪ bangou+hukugousuu　① (bangou+hukugousuu)
② i+hukugousuu　③ (i+hukugousuu)
④ hukugousuu+26　⑤ (hukugousuu+26)

チの解答群

⓪ 25　① 26　② bangou　③ hukugousuu

合同式

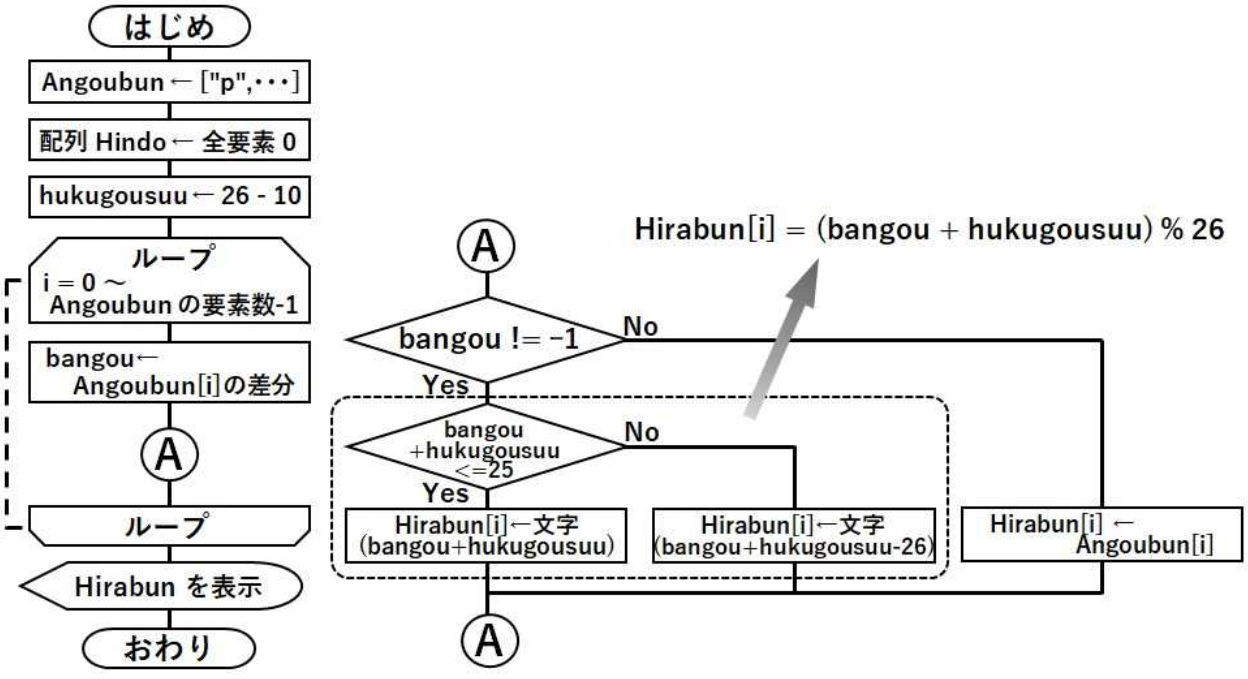
☞ FocusGold 数学ⅠA P.544 参照

例) 10 ≡ 4 (mod 3)
13 ≡ -2 (mod 5)

26を法として合同な値

【算術演算子】

	DNCL	Python
加算	＋	＋
減算	－	－
乗算	＊	＊
除算	／	／
切捨除算	÷	//
剰余算	%	%
べき乗	**	**



【答】タ.1：(bangou + hukugousuu)　チ.1：26