プログラミング基礎２及び演習 期末試験

注意

1. 問題は全部で 11 問ある
2. 問題中のプログラムは全て Python プログラムである
3. 選択肢から選ぶ問題では，同じ選択肢を何回選んでもよい
4. どの問題も，正解はひとつとは限らない

問 1　以下の関数 sort3(a, b, c) は，3 つの数 a, b, c を大きい順（降順）に並べ替えたリストを返す関数である．例えば sort3(1, 2, 3) は [3, 2, 1] を返し，sort3(-5, 10, -5) は [10, -5, -5] を返す．空欄 (1) から (6) に当てはまるリストを書け．

```
def sort3(a, b, c):
    if a >= b and a >= c:
        if b >= c:
            return (1)
        else:
            return (2)
    elif b >= a and b >= c:
        if a >= c:
            return (3)
        else:
            return (4)
    else:
        if a >= b:
            return (5)
        else:
            return (6)
```

問 2　以下の関数 sort3_simple(a, b, c) も，問 1 の sort3 と同じく 3 つの数 a, b, c を大きい順に並べ替えたリストを返す関数である．空欄 (1) から (5) に当てはまるものを以下の選択肢 ①〜⑨ から選べ．x, y = y, x という文は変数 x と y の値を入れ替えることに注意せよ．

```
def sort3_simple(a, b, c):
    if a < b:
        (1)
    if (2):
        (3)
    if (4):
        (5)
    return [a, b, c]
```

① a < b
② b < a
③ a < c
④ c < a
⑤ b < c
⑥ c < b
⑦ a, b = b, a
⑧ a, c = c, a
⑨ b, c = c, b

問 3　以下の triangle_type(a, b, c) は，3 つの数 a, b, c を受け取って，3 辺の長さがそれぞれ a，b，c であるような三角形が存在するか，また，存在する場合はどのような三角形かを表示する関数である．プログラム中の sort3(a, b, c) は問 1 と同じく，a，b，c を大きい順に並べたリストを返す関数である．print 関数による表示結果が常に正しくなるように，空欄 (1) から (5) に当てはまる変数名や条件式を，空白を含めて**それぞれの解答欄の制限字数（= マスの数）以内で**書け．なお，ある頂角が 180° となる場合は三角形ではないとせよ．

```
def triangle_type(a, b, c):
    p, q, r = sort3(a, b, c)
    if (1) <= 0 or (2):
        print("そのような三角形は存在しません")
    elif (3):
        print("正三角形です")
    elif (4):
        print("二等辺三角形です")
    elif (5):
        print("直角三角形です")
    else:
        print("二等辺三角形でも直角三角形でもない")
```

問 4　以下のプログラムを実行したときに表示される数値を書け．

```
x = [4, 3, 5, 7, 9, 10, 12, 8]
s = 0
for i in range(len(x)):
    if x[i] % 2 == 0:
        s += x[i]
    if x[i] % 6 == 0:
        break
print(s)
```

問 5　以下のプログラムを実行したときに表示されるものを書け．

```
def f(xs):
    out = []
    for x in xs:
        if type(x) == list:
            out = f(x) + out
        else:
            out = [x] + out
    return out

xs = [1, 2, [3, 4], [5, [6, 7], [8, 9, 10]], [11, [12], 13]]
print(f(xs))
```

問 6　以下の関数 search(s, t) は文字列 s の部分文字列（s の中の連続するいくつかの文字からなる部分）として文字列 t が含まれるとき True，そうでないとき False を返す関数を実装しようとしたものである．例えば search("cat", "ca") や search("cat", "at") は True を返し，search("cat", "ate") は False を返すようにしたい．しかし，このプログラムは一部間違っている．1 行だけ修正して正しく動くようにしたい．どの行をどう修正すればよいか答えよ．修正する際にインデント（行の最初の空白）は変えないものとして，修正結果はインデント部分を含めずに書け．

行番号

```
 1:  def search(s, t):
 2:      for i in range(len(s)):
 3:          found = True
 4:          for j in range(len(t)):
 5:              if s[i+j] != t[j]:
 6:                  found = False
 7:                  break
 8:          if found:
 9:              return True
10:      return False
```

問 7　以下は，多項式関数の定積分を計算する関数 integral(f, a, b) の実装である．引数 f, a, b はそれぞれ以下を表す:

- f：積分の対象となる多項式関数
- a：積分範囲の下端
- b：積分範囲の上端

引数 f は，被積分関数

$$f(x) = c_n x^n + c_{n-1}x^{n-1} + \cdots c_1 x + c_0 \ \ (c_n, c_{n-1}, \ldots, c_1, c_0 \in \mathbb{R})$$

を，長さ $n+1$ の実数（浮動小数点数）のリスト $[c_0, c_1, \ldots, c_{n-1}, c_n]$ として表す（順番に注意）．例えば $f(x) = 4x^3 - 2x + 1$ はリスト [1, -2, 0, 4] で表される．積分範囲について a ≤ b であることは仮定してよい．例えば $\int_{-1}^{1} x^3 \mathrm{d}x$ は integral([0, 0, 0, 1], -1, 1) で計算できる．プログラム中の空欄 (1), (2) に当てはまる Python の式を書け．

```
def integral(f, a, b):
    d = 0
    s = 0
    for c in f:
        d += (1)
        s += (2)

    return s
```

問 8　以下のような，食品と，それに多く含まれる栄養素のペアのリスト pairs がある:

```
pairs = [("レモン", "ビタミンC"), ("レバー", "ビタミンA"),
         ("牛肉", "たんぱく質"), ("にんじん", "ビタミンA"),
         ("鶏肉", "たんぱく質"), ("赤ピーマン", "ビタミンC"),
         ...]
```

このようなリストを入力し，栄養素をキーとして，それを多く含む食品のリストを値とする以下のような辞書を返す関数 make_dict(pairs) を実装したい：

```
make_dict(pairs)
--> {"ビタミンC" : ["レモン", "赤ピーマン", ...],
     "ビタミンA" : ["レバー", "にんじん", ...],
     "たんぱく質" : ["牛肉", "鶏肉", ...],
     ...}
```

プログラム中の空欄 (1) から (6) に当てはまるものを以下の選択肢①〜⑯から選び，プログラムを完成させよ.

```
def make_dict(pairs):
    (1)
    for p in pairs:
        (2)
        (3)
        (4):
            (5)
        (6)
    return d
```

① `d = ""`　⑨ `if k not in d`
② `d = []`　⑬ `d[k, v]`
③ `d = {}`　⑭ `d{k}[v]`
④ `k = p[0]`　⑩ `d[k] = ""`
⑤ `k = p[1]`　⑪ `d[k] = []`
⑥ `v = p[0]`　⑫ `d[k] = {}`
⑦ `v = p[1]`　⑮ `d[k] += v`
⑧ `if k in d`　⑯ `d[k].append(v)`

問 9　正の整数 m，n の最大公約数を求める関数 gcd(m, n) を実装したい．プログラム中の空欄 (1) から (6) に，以下の選択肢①〜⑪ から当てはまるものを選び，完成させよ.（プログラムの効率は悪い.）

```
def sub(m, n):
    (1)
    if (2):
        return (3)
    return (4)

def gcd(m, n):
    if m < n:
        (5)
    return (6)
```

① `m`
② `n`
④ `m += n`
⑤ `m -= n`
⑥ `m *= n`
⑦ `gcd(m, n)`
⑧ `sub(m, n)`
⑨ `m == 0`
⑩ `n == 0`
⑪ `m, n = n, m`

問 10　以下の関数 f(m, n) と g(m, n) について答えよ．

(1) 任意の整数 m, n > 0 に対して，f(m, n) と g(m, n) が同じ値を返すように，関数 g(m, n) の定義の空欄 [(1)] を 6 文字以内で埋めよ．演算子 // は整数を整数で割ったときの商を返す．例えば 7 // 2 = 3 であり，8 // 2 = 4 である．

(2) f(7, 101) を実行するとき，6 行目と 8 行目の掛け算はそれぞれ合計何回実行されるか答えよ．

行番号

```
 1:  def f(m, n):
 2:      p = m
 3:      q = 1
 4:      while n > 0:
 5:          if n % 2 == 1:
 6:              q *= p
 7:          n = n // 2
 8:          p = p * p
 9:      return q
10:
11:  def g(m, n):
12:      return [ (1) ]
```

問 11　何人かの人がいて，そのうち任意の 2 名を A さん，B さんとすると，A さんは B さんを「好き」「嫌い」「どちらでもない」のうちいずれか一つが必ず成り立つ．それぞれの人は番号で表されているものとする．人々の間の「好き」「嫌い」の関係を以下のような 2 重の辞書 like で表す：

```
like = { 1: { 2: True,  3: False },
         2: { 1: False, 3: False, 5: True },
         ...}
```

この辞書で like[m][n] = True であることは，m 番の人は n 番の人が好きであることを表し，like[m][n] = False であることは，m 番の人は n 番の人が嫌いであることを表す．また，like[m] にキーとして n が含まれない場合は，m 番の人は n 番の人が好きでも嫌いでもない（「どちらでもない」）ことを表す．そもそも like に m がキーとして存在しない場合は，m 番の人は全ての人が好きでも嫌いでもないことを表す．また，全ての人は自分自身が好きでも嫌いでもないとする．

例えば，上の例は 1 番の人は 2 番の人が好きで 3 番の人が嫌いであることや，2 番の人は 1 番の人が嫌いで 4 番の人が好きでも嫌いでもないことを表している．

ここで，ある 3 人 A, B, C の間での

A は B が好きで，B は C が好きで，C は A が好き

という「好き」が循環する関係を「3-好きサイクル」と呼ぶことにする．

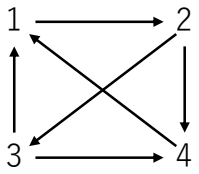
与えられた 2 重の辞書で表される人間関係の中に「3-好きサイクル」がいくつあるか数えたい．その際，

「A は B が好きで，B は C が好きで，C は A が好き」と
「B は A が好きで，A は C が好きで，C は B が好き」は異なる「3-好きサイクル」

である．一方で，

「A は B が好きで，B は C が好きで，C は A が好き」と
「B は C が好きで，C は A が好きで，A は B が好き」は同一の「3-好きサイクル」

だと考える．例えば m 番の人が n 番の人を好きであることを m から n への矢印で表すならば，下図のような人間関係においては 2 つの「3-好きサイクル」が存在する（$1 \to 2 \to 3 \to 1$ および $1 \to 2 \to 4 \to 1$）：



以下のプログラムの空欄 (1) から (10) に，以下の選択肢①〜⑱ から当てはまるものを選び，2 重の辞書 like で表される人間関係の中の「3-好きサイクル」の数を返す関数 count_cycle(like) を完成させよ．

```
def count_cycle(like):
    count = 0
    for a in (1):
        for b in (2):
            if (3) and (4):
                for c in (5):
                    if (6) and (7) and (8) and (9):
                        count += 1
    return (10)
```

① like.keys()
② like[a][b]
③ like[b][c]
④ like[c][a]
⑤ a in like
⑥ b in like
⑦ c in like
⑧ a in like[b]
⑨ a in like[c]
⑩ like[a].keys()
⑪ like[b].keys()
⑫ like[c].keys()
⑬ count
⑭ count / 2
⑮ count / 3
⑯ count / 4
⑰ count / 5
⑱ count / 6