Midterm Exam

2019.11.19 (Tuesday) 09:10 – 12:00

1. **[Python Basics]** In each of the following questions, you are asked to show what will be printed out. If there is an error, please explain why it is an error. (12%)

(a)
```python
print(9//5, "and", 9/5)
```

(b)
```python
print((5 + 4.0/ 3) * 2 + 7 % 6)
```

(c)
```python
a_, _a, a = 2, 3, 5
x, y = a_ + _a + a, a_ - _a - a
print(x, y)
```

(d)
```python
alpha, beta, x = 0.5, 2, 4
answer *= x ** alpha ** beta
print(answer)
```

(e)
```python
a, b = 3, 6
a, b = b/a, a/b
b, a = a/b, b/a
print(a, b)
```

(f)
```python
s = 3.14
print(int(str(float(s))))
```

(g)
```python
course = 'python123'
course[:6] = '456'
print(course)
```

(h)
```python
s = 'Tom'
s[1] = 'I'
print(s])
```

(i)
```python
print("Time left %03d:%05.3f" % (9, 30.19283))
```

(j)
```python
s, t = "ok", "go"
print(st*2)
```

(k)
```python
s = "Love Python?"
print(s[s.count("o")+s.find("Py"):len(s)-1])
```

(l)
```python
s = "i-love-stat"
print(s[1:-1], s[:-1:-1], s[:])
```

2. **[Conditionals & Loops]** In each of the following questions, you are asked to show what will be printed out. If there is an error, please explain why it is an error. (14%)

(a)
```
time, money = 0, -1
print(not money or not time)
```

(b)
```
n, i, check = 31, 2, True
while i < n and check:
    check = ((n % i) == 1)
    i += i
print(check)
```

(c)
```
s, t = "Happy!", []
i = 1
while i < len(s):
    t.append(s[:i])
    i += 2
print(t)
```

(d)
```
x, y = 0, 10
while x != y:
    x, y = x + 2, y - 1
print(x, y)
```

(e)
```
a, b, i = 1, 1, 1
while i < 7:
    a, b, i = b, a + b, i + i
print(a)
```

(f)
```
a, b = 84, 48
while a:
    a, b = b, a % b
print(a)
```

(g)
```
x, a, out, y = 66, 2, "", 1
while x >= 2:
    if x % a != 0:
        a += 1
    else:
        out += " * " + str(a)
        x //= a
        y *= a
print(out[3:], "=", y)
```

3. **[List]** In each of the following questions, you are asked to show what will be printed out. If there is an error, please explain why it is an error. (11%)

(a)
```
d1, d2 = ["Taiwan", "Win", 2020], "2010"
d1[2], d2[2] = d2, 2
print(d1, d2)
```

(b)
```
score = [90, 60, 30, 80]
print((min(score)*max(score)+len(score))/sum(score))
```

(c)
```
game = ["rock", "paper"]
game += game + "scissor"
print(game)
```

(d)
```
n = ["1", "3", "5", "2", "4", "6"]
print(n[-1].join(n[2:4]) * 2)
```

(e)
```
notes = ["Do", "Re", "Mi"]
print(notes[2][0] == "M")  # True or False
```

(f)
```
n = [1]
n.extend(n)
n.append(0)
n.append([0])
print(n)
```

(g)
```
data = "Ada Bob Cat"
print("->".join(data.split()+["Dog"]))
```

(h)
```
v = [1, 2, 3, 4]
p = v
p[2] = 5
print(p == v)
```

(i)
```
x = [5, 3, 2, 2, 6, 1]
x = sorted(x)
x.remove(2)
x.insert(2, 2)
x.pop()
print(x)
```

(j)
```
days = [['Mon','Oct',28], ['Wed','Nov',13], ['Fri','Dec',6]]
print(days[1][0], days[1:], days[2][2][0])
```

(k)
```
i, city = 0, ['Tainan', 'Taipei', 'Taichung']
while i < len(city) - 1:
    if city[i] < city[i+1]:
        city[i-1], city[i+1] = city[i+1], city[i-1]
    i += 1
print(city)
```

4. Below is a Python program that assumes an initialized list `lst` and an initialized integer num are given, and **searches for the last number in list `lst` that is greater than or equal to num**. It returns the index of that number, or `-1` if there is no such number. For example, when the function is called with `[100, 45, 12, 24]` for `lst` and `40` for num, it should return `1`. This is because 45 is the last item in the list that is greater than or equal to `40`. Complete the missing parts of the following code. (16%)

```
# Initialize lst and num here
i, last_index = 0, -1
while i < [    (a)    ]:
    if [    (b)    ]:
        last_index = [  (c)  ]
    i = i + 1
print(last_index)
```

What will be printed out if we have the following initializations?

Initialization	What will be printed out?

`lst = [10, 20, 30, 11, 13]`
`num = 14`
	(d)

`lst = []`
`num = 1`
	(e)

How many times would the `while` loop iterate if the initialization is as below?

Initialization	How many times do the loop iterate?

`lst = list(range(1,100))`
`num = 50`
	(f)

(g) We could **search the list backwards**, looking for an integer that is greater than or equal to num, and return its index **as soon as we can find one**. Write another Python program that **assumes `lst` and num are given** and outputs the same result as above mentioned for the same input, but **works using fewer times of loop iterations**. In other words, your program should output 45 given `lst=[100,45,12,24]` and num=`40`, output index value same as **(d)** given `lst=[10,20,30,11,13]` and num=14, and output index value same as **(e)** given `lst=[]` and num=1.

5. Please read the following code, and answer what will be printed. (8%)

```
# Initialize a and b here
carry, result = 0, ""
i, j = len(a)-1, len(b)-1
while i >=0 or j >= 0:
    if i >= 0:
        carry += int(a[i])
    if j >= 0:
        carry += int(b[j])
    carry, remainder = carry//2, carry%2
    result = str(remainder) + result
    i -= 1
    j -= 1
if carry:
    print(str(carry) + result)
```

Initialization for variables a, b	**Printed?**

`a = "11"`
`b = "1"`
	(a)

`a = "100"`
`b = "1101"`
	(b)

6. Write a program to meet the following three requirements. (15%)
   (1) Allow the user to input a positive integer number x, where x must be larger than 1. Your program should ask the user to input again if not a positive integer larger than 1.
   (2) Check if x is in **Fibonacci sequence**:
   if it is, print what number is it (費氏數列中的第幾個數); if not, print “not”.
   Notice that the elements of Fibonacci sequence are: 0, 1, 1, 2, 3, …, etc.
   (3) Find the x-th **Prime** number. Note: the first Prime number is 2.

7. Write a program to find *Perfect numbers* from 2 to n, where n is input by the user. According to Wikipedia : In the number theory, a perfect number is a positive integer that is equal to the sum of its proper positive divisors. That said, the sum of its positive divisors excluding the number itself. Equivalently, a perfect number is a number that is half the sum of all of its positive divisors (including itself). For example, the first perfect number is 6, because 1, 2, and 3 are its proper positive divisors, and 1 + 2 + 3 = 6. Equivalently, the number 6 is equal to half the sum of all its positive divisors: (1 + 2 + 3 + 6) / 2 = 6. The next perfect number is 28 = 1 + 2 + 4 + 7 + 14. Your program is required to accept an input as n, and output all of the perfect numbers from 2 to n in a list. An example is shown as below. (12%)

Example 1	Example 2

`Input the range number: 1000`
`Perfect numbers: [6, 28, 496]`
	`Input the range number: 10000`
	`Perfect numbers: [6, 28, 496, 8128]`

8. Please write a program with nested `while` loops (e.g., `while` in `while` in `while`) to generate and print a 9×9 multiplication table in a particular order, **as exactly shown in the right figure**. Note that if you totally use `print()` (i.e., without using while loops) to generate the multiplication table, you will get only 1%. (12%)

```
1 x 1 = 1        2 x 1 = 2        3 x 1 = 3
1 x 2 = 2        2 x 2 = 4        3 x 2 = 6
1 x 3 = 3        2 x 3 = 6        3 x 3 = 9
1 x 4 = 4        2 x 4 = 8        3 x 4 = 12
1 x 5 = 5        2 x 5 = 10       3 x 5 = 15
1 x 6 = 6        2 x 6 = 12       3 x 6 = 18
1 x 7 = 7        2 x 7 = 14       3 x 7 = 21
1 x 8 = 8        2 x 8 = 16       3 x 8 = 24
1 x 9 = 9        2 x 9 = 18       3 x 9 = 27

6 x 1 = 6        5 x 1 = 5        4 x 1 = 4
6 x 2 = 12       5 x 2 = 10       4 x 2 = 8
6 x 3 = 18       5 x 3 = 15       4 x 3 = 12
6 x 4 = 24       5 x 4 = 20       4 x 4 = 16
6 x 5 = 30       5 x 5 = 25       4 x 5 = 20
6 x 6 = 36       5 x 6 = 30       4 x 6 = 24
6 x 7 = 42       5 x 7 = 35       4 x 7 = 28
6 x 8 = 48       5 x 8 = 40       4 x 8 = 32
6 x 9 = 54       5 x 9 = 45       4 x 9 = 36

7 x 1 = 7        8 x 1 = 8        9 x 1 = 9
7 x 2 = 14       8 x 2 = 16       9 x 2 = 18
7 x 3 = 21       8 x 3 = 24       9 x 3 = 27
7 x 4 = 28       8 x 4 = 32       9 x 4 = 36
7 x 5 = 35       8 x 5 = 40       9 x 5 = 45
7 x 6 = 42       8 x 6 = 48       9 x 6 = 54
7 x 7 = 49       8 x 7 = 56       9 x 7 = 63
7 x 8 = 56       8 x 8 = 64       9 x 8 = 72
7 x 9 = 63       8 x 9 = 72       9 x 9 = 81
```

9. Given a sorted integer list, where the range of elements are **[lower, upper]** **inclusive**, write a program that can **return its missing ranges**. For example, given `[0, 1, 3, 50, 75]`, lower = 0 and upper = 99, return `["2", "4->49", "51->74", "76->99"]`. Sample input and output are illustrated as below. (10%)

```
c:\workspace>python midterm9.py
Input list: 0 1 3 50 75
Input low: 0
Input high: 99
['2', '4->49', '51->74', '76->99']

c:\workspace>python midterm9.py
Input list: 0 33 66 88
Input low: 11
Input high: 77
['1->32', '34->65', '67->87']

c:\workspace>python midterm9.py
Input list: 5 6 7 21 22 23 24 25 52
Input low: -10
Input high: 66
['-10->4', '8->20', '26->51', '53->66']
```