CITY UNIVERSITY OF HONG KONG

Course code & title: CS1302 Introduction to Computer Programming
Session : Semester B 2021/22
Time allowed : Two hours

This paper has 11 pages (including this cover page).

This paper consists of 3 parts. Answer ALL the questions in the space provided within each question.

Question	Part 1 (10 marks)	Part 2 (30 marks)	Part 3 (60 marks)	**Total (100)**
Marks				

*This is a **closed-book** examination.*

No materials or aids are allowed during the whole examination. If any unauthorized materials or aids are found on a student during the examination, the student will be subject to disciplinary action.

Academic Honesty

I pledge that the answers in this examination are my own and that I will not seek or obtain an unfair advantage in producing these answers. Specifically,

- *I will not plagiarize (copy without citation) from any source;*
- *I will not communicate or attempt to communicate with any other person during the examination; neither will I give or attempt to give assistance to another student taking the examination; and*
- *I will use only approved devices (e.g., calculators) and/or approved device models.*
- *I understand that any act of academic dishonesty can lead to disciplinary action.*

I pledge to follow the Rules on Academic Honesty and understand that violations may lead to severe penalties.

Student ID:
Name:

CS Departmental Hotline (phone, whatsapp, wechat)
+852 6375 3293

Part 1 Basic knowledge (10 marks)

Complete the blanks with the provided options. Note that each option may be used multiple times, and not all the options will be used (there're 12 options but only 10 blanks) (1 mark for each blank).

Options:

list	tuple	set
dictionary	iterations	generator expression
recursion	yield expression	exception
generator	base cases	recursive call

- A __(1)__ is a function that calls itself. To avoid the infinite call, each __(2)__ to the function itself must get closer to some __(3)__ that terminate the call. It is always possible to re-implement such a function using __(4)__ without calling the function itself.

- A __(5)__ is a function that can generate values in a sequence one at a time when the next value is needed. Such a function can be written as a __(6)__ or returned by a function that has a __(7)__. It raises an __(8)__ if it cannot return the next value.

- A __(9)__ is a collection which is ordered and unchangeable, and allows duplicate members.

- A __(10)__ is a collection which is unordered, unindexed, and doesn't allow duplicate members.

Your answers:
(1)_____
(2)_____
(3)_____
(4)_____
(5)_____
(6)_____
(7)_____
(8)_____
(9)_____
(10)_____

Part 2 Code Understanding (30 marks)

Each part below contains a code segment written in Python. Please write down the output of the code (1 mark for each correct print output).

(1)

```
print(2**(3**2))
print(2**3**2)
print(24 // 6 % 3)
print(24 // 4 // 2)
```

Output

(2)

```
x1 = complex(1, 2)
x2 = complex(1, 3)
print([x1, x2] == [x2, x1])
print({x1, x2} == {x2, x1})
print(x1 + x2 is complex(2, 5))

x1, x2, x2 = x2, x2, x1
print(x1,x2)
```

(3)

```
for f in ["chicken", "fish"]:
    for m in ["steam", "roast"]:
        print(m, f)
```

(4)

```
i = 5
while i > 0:
  print(i)
  i += 1
  if i == 8:
    break
else:
  print(i)
```

(5)

```
def test(i,j):
  if(i==0):
    return j
  else:
    return test(i-1,i+j)

print(test(4,7))
```

(6)

```
total = 0

def foo(x):
    total = x +1
    return total+1

print(foo(total))
print(total)
```

(7)

```
def foo(x):
    x = ['def', 'abc']
    return id(x)

q = ['abc', 'def']
print(id(q) == foo(q))
```

(8)

```
x = 5
print(1 < x < 10)
print(10 < x < 20 )
print(x < 10 < x*10 < 100)
print(10 > x <= 9)
print(5 == x > 4)
```

(9)

```
a, b, c, d, e, f = 0, 5, 12, 0, 15, 15
exp1 = a <= b < c > d is not e is f
exp2 = a is d > f is not c

print(exp1)
print(exp2)
```

(10)

```
def foo():
    x = 10
    while True:
        x -= 1
        y = yield x*2
        x += y or 1
    else:
        print("stopped")

g = foo()
print(next(g))
print(g.send(5))
print(next(g))
print(g.send(False))
```

Part 3 Coding (60 marks)

Please complete the following programming questions (8 coding questions).

1. (6 marks) Define a function sorted_and_dedupped() that
- takes a sequence as an argument, and
- returns a sorted list of distinct values of the sequence.

For example:

Test	Result
sorted_and_dedupped([3,3,2,2,1,1])	[1, 2, 3]
sorted_and_dedupped([1,2,7,3,4,7,2])	[1, 2, 3, 4, 7]

Answer:

```
def sorted_and_dedupped(seq):
#Add your code below
```

2. (6 marks) Define a function power_sequence() that
- takes a floating point number p and a non-negative integer stop, and
- returns a generator that generates x^p for non-negative integer x from 0 up to and excluding stop.

For example:

Test	Result
gen = power_sequence(1,10) print([next(gen), *gen])	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
gen = power_sequence(2,11) print([next(gen), *gen])	[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
gen = power_sequence(0.5,11) print([round(x,2) for x in gen])	[0.0, 1.0, 1.41, 1.73, 2.0, 2.24, 2.45, 2.65, 2.83, 3.0, 3.16]

Answer:

```
def power_sequence(p,stop):
#Add your code below
```

3. (6 marks) Complete the function snake(width, height) which inputs the size of a rectangular zone: number of columns and number of rows, and paints a snake inside the zone.

For example:

Test	Result
snake(8, 5)	******** * ******** * ********

snake(6, 10)	******
	     *
	******
	*
	******
	     *
	******
	*
	******
	     *

Answer:

```
def snake(width, height):
# Add your code below
```

4. (6 marks) Catalan number C_n of order n is an integer defined by the following recurrence equation:

$$C_0 = 1$$

$$C_n = \sum_{i=0}^{n-1} C_i C_{(n-1)-i} \quad \text{for } n > 0$$

$$= C_0 C_{n-1} + C_1 C_{n-2} + \cdots + C_{n-1} C_0.$$

Catalan numbers have many engineering applications including computational geometry and cryptography. It counts the number of expressions containing n pairs of parentheses which are correctly matched. For n=3, there are C_n=5 such expressions: ((())), ()(()), ()()(), (())(), (()()).

Define a recursion catalan that takes a non-negative integer order n and returns the Catalan number of order n. For this question, you may implement the recurrence equation directly without worrying about redundant computations.

For example,

Test	Result
print([catalan(n) for n in range(5)])	[1, 1, 2, 5, 14]

Answer:

```
def catalan(n):
#Add your code below
```

5. (6 marks) Define a function arithmetic_geometric_mean_sequence() that
- takes two floating point numbers x and y and
- return a generator that generates the tuple (a_n,g_n) where

$$a_0 = x, g_0 = y$$

$$a_n = \frac{a_{n-1} + g_{n-1}}{2} \quad \text{for } n > 0$$

$$g_n = \sqrt{a_{n-1}g_{n-1}}$$

For example:

Test	Output
`result = arithmetic_geometric_mean_sequence(6,24)` `print([next(result) for i in range(2)])`	`[(6, 24), (15.0, 12.0)])`
`result = arithmetic_geometric_mean_sequence(2,8)` `print([next(result) for i in range(2)])`	`[(2, 8), (5.0, 4.0)]`

Answer:

```
def arithmetic_geometric_mean_sequence(x,y):
#Add your code below
```

6. (6 marks) Define a function has_duplicates that

- takes an argument d of type dict, and
- returns True if d has duplicate values for different keys, and returns False otherwise.

For example,

Test	Output
`d = dict({1: 'a', 2: 'b'})` `print(has_duplicates(d))`	`False`
`d = dict({1: True, 2: 1})` `print(has_duplicates(d))`	`True`

Answer:

```
def has_duplicates(d):
#Add your code below
```

7. (a) (4 marks)

Define a function rand_sm_big to provide a generator that yields lists of two random numbers in the range [0,1) in ascending order.

Hint: you may need random.random() function which returns a random number

For example,

Test	Output
r1 = rand_sm_big() print(next(r1)) print(next(r1))	[0.07068996502155289, 0.939268345476326] [0.4125282017784728, 0.934907226205321]

Answer:

```
import random

def rand_sm_big():
#Add your code below
```

(b) (8 marks)

Revise the function rand_sm_big such that we can use send() to input a list of source numbers to pick from. We can also input an empty list to restore to the original mode as in part (a), i.e. picking random numbers in the range[0,1).

Hint: you may need random.choice() function which randomly selects an element from the input argument.

For example,

Test	Sample Output
r = rand_sm_big() print(next(r)) print(r.send([100,30,4,4,6,100]))	[0.07094419796353435, 0.9776256953373558] [4, 6]
r = rand_sm_big() print(next(r)) print(r.send([100,30,4,4,6,100])) print(next(r)) print(next(r)) print(r.send([])) print(next(r))	[0.2078840656984643, 0.7873297571202036] [4, 100] [6, 6] [4, 100] [0.02220297757279588, 0.28118288404392766] [0.10353478564782614, 0.5909971569169883]

Answer:

```
import random

def rand_sm_big():
#Add your code below
```

8. Students have answered a short quiz, with typing mistakes as usual. Here we have a set of strings wDict of all English words that are correctly spelt. We assume students' answers and the strings in wDict are in lowercase.

By matching with wDict, we can identify misspelt words from students' answers. As a further analysis, we can check the common misspelt words from any pair of two students.

(a) (4 marks) Define a function misspelt that
- takes an argument student_answer of type str that holds a student's answer, and
- return a set of misspelt words from student_answer

For example,

Test	Output
`print(misspelt("lisst is immutable"))`	`{'lisst'}`
`print(misspelt("1ist is a imutable"))`	`{'imutable', '1ist'}`
`print(misspelt("lisst is immutoble in pyphon"))`	`{'pyphon', 'lisst', 'immutoble'}`
`print(misspelt("a lisst in python is immutoble"))`	`{'lisst', 'immutoble'}`

Answer:

```
import nltk
nltk.download('words')
from nltk.corpus import words
wDict = list(words.words())

def misspelt(student_answer)
#Add your code below
```

(b) (8 marks) Define a function analysis that
- takes an argument answers of type dict, in which each entry maps a student's ID to his answer, and
- returns the result in a dict type that contains the common misspelt words between all pairs of students. Those student pairs that do not have common misspelt words are excluded. Each entry in the result should map a tuple of two students' IDs in ascending order to the set of their common misspelt words.

For example,

Test	Sample Output

`analysis(`
`  {50111023: "lisst is immutable",`
`  50123888: "1ist is a imutable",`
`  51992299: "lisst is immutoble in pyphon",`
`  53332233: "a lisst in python is immutoble"}`
`)`
	`{(50111023, 51992299): {'lisst'},`
	`(50111023, 53332233): {'lisst'},`
	`(51992299, 53332233): {'immutoble', 'lisst'}}`

Answer:

```
def analysis(answers)
#Add your code below
```


-- END OF PAPER --