Week 4 Tuples, Lists and other Iterables

For Parts 1 to 3, try to come up with the answer without using IDLE/Python first. If there is an error, specify the cause and type of the error. Then type the expressions into IDLE to verify your answers. The objective is for you to understand why and how they work.

Part 1 Tuples

Code	**Output**

tup_a = (10, 11, 12, 13)
print(tup_a)
tup_b = (“CS”, 1010)
print(tup_b)
tup_c = tup_a + tup_b
print(tup_c)
print(len(tup_c))	

print(11 in tup_a)
print(14 in tup_b)
print(“C” in tup_c)
print(tup_b[1])
tup_d = tup_b[0]*4
print(tup_d)
print(tup_b[1] * 4)	

tup_e = tup_d[1:]
print(tup_e)
tup_f = tup_d[::-1]
print(tup_f)
tup_g = tup_d[1:-1:2]
print(tup_g)
tup_h = tup_d[-1:6:-2]
print(tup_h)	

tup_i = (1)
print(tup_i)
tup_j = (1,)
print(tup_j)
print(tup_i * 4)
print(tup_j * 4)	

print(min(tup_a))
print(max(tup_a))
print(min(tup_c))
print(max(tup_c))
print(min(tup_e))
print(max(tup_e))	

`for i in tup_b:`
`  print(i)`	

`for i in range(5):`
`  print(i)`	

`for i in range(2,5):`
`  print(i)`	

`for i in range(2,5,2):`
`  print(i)`	

`for i in range(5,1,-1):`
`  print(i)`	

`for i in range(5,6,-1):`
`  print(i)`	

Part 2 Lists

Code	**Output**

`lst_a = [“CS”, 1010]`
`print(lst_a)`
`lst_b = [“E”,(“is”, “easy”)]`
`print(lst_b)`
`lst_c = lst_a + lst_b`
`print(lst_c)`	

`tup_a = (“CS”, 1010)`
`tup_a[1] = 2030`
`lst_a[1] = 2030`
`print(lst_a)`	

`lst_a.append(“E”)`
`print(lst_a)`
`lst_a.extend(“easy”)`
`print(lst_a)`	

`cpy_b = lst_b[:]`
`print(cpy_b)`
`cpy_b[1] = “is hard”`
`print(cpy_b)`
`print(lst_b)`	

lst_d = [1, [2], 3]
cpy_d = lst_d[:]
print(cpy_d)
print(lst_d)
lst_d[1][0] = 9
print(cpy_d)
print(lst_d)	

print(lst_d == cpy_d)
print(lst_d is cpy_d)
print(lst_d[1] == cpy_d[1])
print(lst_d[1] is cpy_d[1])	

Part 3 Mutation

Code	**Output**

x = 1
y = x
x = 2
print(x)
print(y)	

lstx = [1,2,3]
lsty = lstx
lstx[0] = 999
print(lstx)
print(lsty)	

What is the difference between the two code snippets above?

Code	**Output**

a = 4
 	
def foo(x):
    x = x * 2
    print(x)
 	
print(a)
foo(a)
print(a)	

lsta = [1,2,3]
 	
def foo2(lst):
    lst[0] = lst[0]*2
    lst[1] = lst[1]*2
    print(lst)	

`print(lsta)`
`foo2(lsta)`
`print(lsta)`	

What is the difference between the two code snippets above?

Part 4 Course Schedule

We can add mixed data types into tuples and/or lists. In this exercise, you are hired by NUS to improve the EduRec system because of the many complaints received. You are provided with an implementation for courses as follows:

```
def make_course(code, units):
  return (code, units)

def make_units(lec, tut, lab, hw, prep):
  return (lec, tut, lab, hw, prep)

def get_course_code(course):
  return course[0]

def get_course_units(course):
  return course[1][0] + course[1][1] + course[1][2] + course[1][3] + course[1][4]
```

Abstraction is the removal of unnecessary details from the problem we want to solve. When these details are 'abstracted away', we can have a clearer view of the problem. You are to respect abstraction barriers in this question. One way to think of it is to assume you do not know what statements are in the inner body of the above functions. For instance, to obtain the course code from a course `course`, you should not know that you can index `course` at position 0. Instead, you should make use of the function `get_module_code(course)`.

A. Write a function `make_empty_schedule()` that returns an empty schedule of courses.
B. Write a function `add_course(course, schedule)` that returns a new schedule with the added course.
C. Write a function `total_scheduled_units(schedule)` that returns the total number of units of all courses in the schedule.
D. Write a function `drop_course(course, schedule)` that returns a new schedule without the specified course.
E. Write a function `valid_schedule(schedule, max_units)` that returns a new schedule with the total number of units less than or equal to `max_units` by removing courses from the specified schedule.