Results

Shang Xiao — 1st Attempt

100%

10
Out of 10 points

130:28:56
Time for this attempt

This assessment has unlimited attempts. Take Now

Attempt History

Results	Points	Score	(Highest score is kept)
Attempt 1	10 of 10	100%	(Highest score)
Attempt 2	0 of 10	0%	

Your Answers:

1 2 / 2 points

What are 2 reasons to use data structures?

Asking whether we should be using data structures is like asking whether you should replace your iPhone 4 with iPhone 12 Pro. The answer is, quoting professor John Park, "not absolutely critical," but it will make your life so much more easier and more convenient.

Imagine that we live in a world without data structures such as lists and tuples. That would be so horrifying because when dealing with a large chunk of data, such as each student's grade in a class, you would then have to key in every single grade one by one and accessing them one by one, which could costs you weeks or months to complete such a simple task; however, this would not happen if you were to know there are much simpler data structures like lists or tuples.

In conclusion, although the information they are producing is the same, the way to organize, store, or access them can vary in so many more different ways. Some are super simple and straightforward, while others are just terribly complex. Data structures make information so much more organized and easier to access.

Another reason is that you can choose the appropriate data structures for specific tasks. You get to choose the data structure based on the characteristics of the task. For example, dictionaries would be the most suitable data structure for a phone book entry. In this scenario, data structures provide you with a link to the most efficient way by connecting between a phone book and a dictionary. Of course, you can also use lists, but that can be very hard to code.

Correct

2 2 / 2 points

You've made a set of associative arrays where array1 holds names and array2 holds favorite foods.

array1 = {"Joe", "Bill", "Sue", "Tamyra", "Handy"}
array2 = {"pizza", "pineapple", "tacos", "spaghetti and meatballs", "chocolate"}

What would happen if "Sue" was removed from array1 but no changes happen to array2? Is this a problem? Why or why not?

Yes this is a problem.

If we look at two original associative arrays, we would rationally assume that:

Joe's favourite food is pizza,

Bill's favourite food is pineapple

Sue's favourite food is tacos,

Tamyra's favourite food is spaghetti and meatballs,

Handy's favourite food is chocolate;

When we remove Sue, the associativity of two arrays has changed, and thus, 'Tamyra' now holds the index value of which 'Sue' was previous holding, and so if we were to call get_food again, we can find that Tamyra's favourite food will no longer be spaghetti and meatballs; instead Tamyra's favourite food now becomes Tacos, which was the favourite food of Sue previously.

Correct

3 2 / 2 points

Given the arrays in the previous example, turn them into a dictionary with name being the key.

```
array1 = {"Joe", "Bill", "Sue", "Tamyra", "Handy"}
array2 = {"pizza", "pineapple", "tacos", "spaghetti and meatballs", "chocolate"}

favourite_foods = {"Joe": "pizza",
                   "Bill": "pineapple",
                   "Sue": "tacos",
                   "Tamyra": "spaghetti and meatballs",
                   "Handy": "chocolate"
                   }

favourite_foods2 = dict(Joe = "pizza", Bill = "pineapple", Sue = "tacos", Tamyra = "spaghetti and meatballs",
                        Handy = "chocolate")

favourite_foods3 = dict([["Joe", "pizza"], ["Bill", "pineapple"], ["Sue","tacos"],
                         ["Tamyra", "spaghetti and meatballs"], ["Handy","chocolate"]])


def main():
    print(favourite_foods)
    print(favourite_foods2)
    print(favourite_foods3)

main()
```

Correct

4 2 / 2 points

By looking at the Python set documentation, pick out two interesting functions on sets and explain them.

The 'add' method, `set.add(elmnt)` adds an element to the set, if the element being added already in the set, the method `add()` does not add the element, for example:

```
numbers = set(["1", "2", "3", "4", "5"])


def add9():
    numbers.add("9")

def add2():
    numbers.add("2")

def printNumbers():
    for i in numbers:
        print (i)

def main():
    add9()
    add2()
    printNumbers()

main()
```

We need to take note that:

The output from printNumbers() does not print the set numbers in its original order, printNumbers prints elements in the set randomlly.

add2() function will not add "2" into the set numbers again because there is already "2" in the set.

The 'remove' method, `set.remove(item)` removes an element to the set, if the element being removed is not in the set, the method `add()` does not remove the element and will return a key error for example:

```
numbers = set(["1", "2", "3", "4", "5"])

def remove3():
    numbers.remove("3")

def printNumbers():
    for i in range(len(numbers)):
        print (i)

    print ("---")

    for c in numbers:
        print (c)

def main():
    remove3()
    printNumbers()

main()
```

Correct

5 2 / 2 points

Describe the inheritance organization if you have a class Person and sub classes Faculty, Student and Staff.

Class in python means you collect a bunch of things that share the same ATTRIBUTE. For example, all person can commonly have their names, addresses, and phone numbers; some particular person has these attributes in common and some other attributes. In this case, each faculty can have their own unique faculty ID, same for students, students ID, and staff, staff ID; however, these unique IDs do not have to be the common attribute for all other persons other than faculty, student, and staff. For example, an elder person does not have any of these IDs.

When we first define class Person, common attributes such as names, addresses, and phone numbers can be coded as three methods. For subclasses Faculty, Student, and Staff, they share the same attributes in class Person. And we do not need to duplicate the code we have written in class Person again. We only need to write the unique code for each of these three subclasses if any attribute differs from the common attributes that class Person has. Also, when we only make changes to sub classes, such as Faculty, the changes we made there will not affect class Person, the changes made in sub class Faculty will just stay in the subclass Faculty and will only be unique to class Faculty.

In another word, Person is a super class of Faculty, Student, and Staff.

We need to take note that in this example, Faculty, Student, Staff is not a subclass of each other because even though they all inherit from Person, they are not part of each other's inheritance. We can usually check this by using the *isinstance* and *issubclass* function.

Correct