6.2 - Week 6 - Applied Practical

Objectives of this Applied Session

The objectives of this tutorial are to ensure we understand:

- Names, namespaces, scopes, and scoping rules.
- Classes, Objects, and inheritance
- Mutable and immutable objects
- Using the SetADT

and we know how to both implement and follow code that combines these concepts.

Namespaces

Perform the following steps:

1. Import the built-in `array` module and display its namespace. Check what the different elements in its namespace do.

2. See the class created in the scaffold file `namespaces.py` and try to print its namespace. What do you see in there? Discuss the results with your tutor.

3. Write a Python program to create an instance of a specified class above and display the namespace of the said instance.

What have you learned about namespaces? Read about the elements you have discovered and discuss with your tutors

Classes and Objects

1. Create an abstract class Student that will be used for Monash University. Think about what kind of variables you want to make as class variables and instance variables.

2. Create a child class for `ClaytonStudent` or `MalaysiaStudent` according to whichever campus you are on. Populate the abstract methods that you created above.

3. Create a `Unit` class that contains details of a subject that a student can take at the University. Note that the unit can be taught at both campuses.

4. Now, create a timetables class that sets up the timetable for a student at a particular campus. Feel free to use your imagination, but make sure to use the methods that you created in the 3 classes above.

The student with the best looking timetable before the end of the class wins! Your tutor choses the winner :)

Mutability

We will learn all about mutability in python in this exercise. Follow along even if you know everything about mutability.

Question 1 *Submitted Aug 30th 2022 at 9:25:14 am*

Observe the following code:

```
print(id('1'))
print(id(1))
x = '1'
print(id(x))
```

What do you think the `id` function does?

- Shows random numbers
- [✓ Shows the `id` of the argument passed in the function]
- Shows the `id` of the variable passed in the function
- Shows the `id` of the string passed in the function

Question 2 *Submitted Aug 30th 2022 at 9:25:31 am*

Is the `id` of the following two items the same?

```
print(id('1'))
x = '1'
print(id(x))
```

- [✓ True]
- False

Question 3 *Submitted Aug 30th 2022 at 9:25:48 am*

Now, let us try and manipulate a string. A string is a list of characters, hence we are able to loop through each character. So, by definition, we should be able to replace a character at a particular index.

```
string = "Rick Astley"
string[0] = "M"
print(string)
```

What happens in the code above?

- ○ It prints `Mick Astley`
- ○ It throws a `ValueError`
- [✓ ● It throws a `TypeError` because strings cannot use the `__setitem__` method]
- ○ It throws a `TypeError` because strings are not really lists

Question 4 *Submitted Aug 30th 2022 at 9:26:05 am*

But wait a minute, can't we just change the string? That surely 'mutates' the string, right?

I could do this:

```
string = "Rick Astley"
string = "Mick Astley"
print("tadaaaa")
```

- ○ The string has mutated!
- ○ You found a loophole in Python, this is groundbreaking!
- [✓ ● The variable now points to a completely different string altogether]
- ○ Nothing has really happened, this is a brand new variable

Question 5 *Submitted Aug 30th 2022 at 9:26:18 am*

Lets perform the same steps as above, but also check the ID of the variable.

```
string = "Rick Astley"
print(id(string))
string = "Mick Astley"
print(id(string))
```

Does the `id` of the string change?

- True
- [✓ False]

Question 6 *Submitted Aug 30th 2022 at 9:26:36 am*

Now, what about lists? Lists are surely mutable, right? Lets check it out

```
my_list = [1,2,3,4]
print(id(my_list))
my_list[2] = 5
print(id(my_list))
```

What is the output of the program?

- Throws an error like before, nothing is mutable
- The program prints the `id` but it changes after the change in item
- [✓ The `id` is printed and it stays the same]

Question 7 *Submitted Aug 30th 2022 at 9:26:46 am*

What do you reckon will happen if we create two variables that point to the same list?

Will their IDs be the same?

- [✓ True]
- False

Question 8 *Submitted Aug 30th 2022 at 9:26:55 am*

Let's test out the theory above:

```
my_list = [1,2,3,4]
my_list_but_better = my_list

print(id(my_list))
print(id(my_list_but_better))
```

Are the IDs the same for both list variables?

[✓ True]

False

Question 9 *Submitted Aug 30th 2022 at 9:27:11 am*

Okay, now answer this without running the code:

```
my_list = [1,2,3,4]
my_list_but_better = [1,2,3,4]

print(id(my_list))
print(id(my_list_but_better))
```

What will be the result of this code above?

Two same lists, so it should be the same ID

Two different lists, IDs should be the same though

[✓ Two different lists with two different IDs]

I have literally no idea what is going on right now, this is contradictory to what I have learned so far and I am really confused and hungry and just want to go to sleep for a very long time

Question 10 *Submitted Aug 30th 2022 at 9:27:22 am*

Alright! Last question and then you're done until next week.

What about objects of classes that you create? Do they always have the same ID if they are pointing to each other?

[✓ True]

○ False