Containers, Aliasing Mutability

Necla Mutlu
nmutlu17@ku.edu.tr

Date: 4th of March

1 Overview

This handout is prepared for KOLT Python Certificate Program. It contains a brief review of this week's topics and exercise questions.
You can download the starter code from here.
You can find the solutions at here after all the sections are conducted.

2 Review

2.1 Simple Functions

Functions are blocks of **organized**, **reusable** code that carry some **specific** tasks. We will talk more about functions in more detail soon, however, we wanted to briefly introduce the most simple form of functions to help you write better Python code. In Python, indentation is used to group different expressions. Recommended amount for an indentation level is 4 spaces.

```
def function_name():
    <expression>
    <expression>
    <expression>
    ...
```

Code snippet 1: Syntax for defining a simple function

2.2 Branching

We might want to control how our program behaves based on different conditions. A *condition* can be any Python expression, Python will evaluate this expression and decide on which `code block` will be executed based on whether this expression has a **falsy** or **truthy** value.

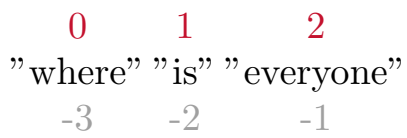
```
if <condition>:
    <expression>
    ...

elif <condition>:
    <expression>
    ...
else:
    <expression>
    ...
```

Code snippet 2: `if-elif-else` statement. You can add as many `elif` blocks as you want. Think about the difference between having `elif` blocks and `if` blocks.

2.3 Lists

In python, we can store data in objects called lists. They are not different than a normal list in the daily life at all. One feature of them is that every element in the list is assigned a room number called index starting from the left most element.

```
words = ["where", "is", "everyone"]
```




```
empty_list = []
letters = ['a', 'b', 'c', 'd']
numbers = [2, 3, 5]
numbers.append(7) # => numbers = [ 2, 3, 5, 7]
numbers.remove(2) # => numbers = [ 3, 5, 7]
print(5 in numbers) # => True
print(len(numbers)) # => 3
numbers.insert(0,9) # => numbers = [9, 3, 5, 7]
```

Another important property of the lists is that we can slice them and extract the parts that we want.

```
y = ['a', 'b', 'c', 'd', 'e', 'f']
y[:3] # => ['a', 'b', 'c']
y[2:] # => ['c', 'd', 'e', 'f']
y[:-1] # => ['a', 'b', 'c', 'd', 'e']
y[:] # => ['a', 'b', 'c', 'd', 'e', 'f']
y[1:5:2] # => ['b', 'd']
y[::3] # => ['a', 'd']
```

2.4 For Loops

For loops generally used to go over the elements in iterables.

```
for <item> in <iterable>:
    <expression>
    <expression>
    ...
```

To go over the each element in a list, sometimes it is more handy to go over indices. To do that, we can use **range()** function, which returns an iterable of numbers from the start to the end. Basically, you can do the same operations on the elements after getting the element at the index specified by using square brackets.

```
range(0,5) # 0, 1, 2, 3, 4 => right end is not inclusive.
range(0,5,2) # 0, 2, 4 => right end is not inclusive.
for i in range(0,8):
    print(i)
mylist = [4,23,12,0,50]
for num in mylist:
    print(num * 3,sep=".")
```

```
for i in range(0,len(mylist)):
    print(mylist[i] * 3,sep=".")
```

2.5 Strings

Textual data in Python is handled with `str` objects, or strings. Strings are immutable sequences of Unicode code points. String literals are written in a variety of ways:

- **Single quotes**: 'allows embedded "double" quotes'
- **Double quotes**: "allows embedded 'single' quotes".
- **Triple quoted**: '''Three single quotes''', """Three double quotes"""

Strings are immutable objects, which means that we cannot modify them. For example, the following code will cause an error:

```
my_string = "hello"
my_string[2] = 'm'
# PYTHON WILL SHOUT AT US!
```

2.6 Functions

Functions are blocks of organized, reusable code that carry some specific tasks.

```
def function_name():
    <expression>
    ...
def function_name(parameter1, parameter2, ...):
    <expression>
    ...
```

Code snippet 3: Defining a function only makes it available. You should call the function to execute.

- *return* statement immediately terminates the function.

```
def function_name(parameter1, parameter2, ...):
    <expression>
    ...
    return value
```

Code snippet 4: Functions implicitly return None if they complete without a return statement.

2.7 Tuples

- Immutable sequence(ordered) of elements.
- Similar to lists, you can use indexing, slicing, and iterate over using for loops.
- Elements cannot be added/removed/changed once the tuple is created.

```
my_tuple = (1, [1, 2],'a')
len(my_tuple) #     3
my_tuple.append(3) # PYTHON WILL AGAIN SHOUT AT US!
(3)  # int 3
(3,) # Be careful about the comma, tuple containing 3
print(my_tuple[0]) # 1 we can use indexing like in the lists.
```

2.8 Sets

- Unordered sequence of unique elements.
- Cannot use indexing/slicing, can iterate with for loops.
- Mutable, add(element), remove(element) methods.

```
ceren = set()
ceren.add('Marco')
ceren.add('Irem')
ceren.add('Sunduz')
gul_sena = {'Gamze', 'Ata', 'Zeynep'}
hasan_can = {'Gamze', 'Berker', 'Cemre'}
ahmet = {'Irem', 'Demet', 'Ekin'}
# intersection &
print(gul_sena.intersection(hasan_can)) # => {'Gamze'}
print(ceren & gul_sena) # => set()
# union |
print(ceren.union(ahmet)) # => {'Ekin', 'Irem', 'Demet',
# 'Marco', 'Sunduz'}
print(hasan_can | ceren | gul_sena | ahmet) # => all names
# difference -
print((gul_sena - hasan_can)) # => {'Zeynep', 'Ata'}
# symmetric_difference
print(ceren.symmetric_difference(ahmet))
# => {'Marco', 'Ekin', 'Sunduz', 'Demet'}}
```

3 Exercises

3.1 Check Subset

Implement a program that checks whether given two sets, one of them is the subset of the other or not. **Input:**

- The length of the first set
- The elements of the first set one by one
- The length of the second set
- The elements of the second set one by one

```
This checks whether one of the two given sets is the subset of the
    other.
Set1 = [1,2,3] Set2[1,2] => YES
Set1 = [1,2,3] Set2[7,2] => NO
Set1 = [1,2,3] Set2[1,2,3] => YES
```

3.2 Remove duplicates

Implement a program which inputs a string and an integer "k" from the user and divides the string into k pieces. After that, your program should remove duplicate letters in those k pieces and output the remaining strings.

1. Input a text and an integer "k" from the user.
2. Write a function which divides a string into "k" pieces and returns a list storing those pieces.
   * You can assume that the user will give a string with length which is divisible by k.
3. Write a function which removes duplicate letters in each element of the given list.
4. Print the resulting string pieces.

Input Text: AABCAAADA
Integer k: 3

```
This program divides string into k pieces and prints each piece
    after removing duplicate letters in it.
"""
AAB => remove one of the A's
CAA => remove one of the A's
ADA => remove one of the A's
"""
Output:
AB
CA
AD
```

3.3 Multiply Elements of Tuple

You will be given n tuples that will have 2 elements. Your program should output the multiplication of the elements in each tuple.

1. First, take "n" ,the number of tuples, from the user.
2. Then for each tuple, you will be given two numbers one by one. Take them from the user as inputs.
3. Create a tuple with the values you get and add them to a list, which will store all the tuples given to you.
4. Go over the list and do the operations.

Input:

- Number of tuples: n =¿ integer
- 2 elements for each tuple =¿ both will be integers

```
This program takes a list of tuples with two elements from user and
    prints the multiplication of the elements in each tuple.
[(2, 3), (4, 5), (6, 7), (2, 8)]
6
20
42
16
```