LAB # 2: Data types, Containers, Input/output, and Operators in Python.

Python Strings

String is sequence of Unicode characters. We can use single quotes or double quotes or even triple quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """. A string in Python consists of a series or sequence of characters - letters, numbers, and special characters. Strings can be indexed - often synonymously called subscripted as well. Similar to C, the first character of a string has the index 0.

Different string syntaxes (simple, double or triple quotes):

```
>>>s = 'Hello, how are you?'
s = "Hi, what's up"
>>>s = '''Hello, # tripling the quotes allows the
how are you''' # string to span more than one line
>>>s = """Hi,
what's up?"""
```

```
In [1]: s = 'Hello, how are you?'

In [2]: s = "Hi, what's up"

In [3]: s = '''Hello, # tripling the quotes allows the
   ...: how are you''' # string to span more than one line

In [4]: s = """Hi,
   ...: what's up?"""

In [5]: print(s)
Hi,
what's up?

In [6]: print(type(s))
<class 'str'>

In [7]:
```

The newline character is \n, and the tab character is \t. Strings are collections like lists. Hence they can be indexed and sliced, using the same syntax and rules.

Indexing:

```
>>>a = "hello"
>>> a[0]
>>> a[1]
```

```
>>> a[-1]
```

(Remember that negative indices correspond to counting from the right end.)

```
In [8]: a = "hello"

In [9]: a[0]
Out[9]: 'h'

In [10]: a[1]
Out[10]: 'e'

In [11]: a[-1]
Out[11]: 'o'
```

```
In [17]: print(a[len(a)-1])
o
```

Slicing:

```
>>> a = "hello, world!"
>>> a[3:6] # 3rd to 6th (excluded) elements: elements 3, 4, 5
>>> a[2:10:2] # Syntax: a[start:stop:step]
>>> a[::3] # every three characters, from beginning to end
```

```
In [20]: a = "hello, world!"

In [21]: a[3:6] # 3rd to 6th (excluded) elements: elements 3, 4, 5
Out[21]: 'lo,'

In [22]: a[2:10:2] # Syntax: a[start:stop:step]
Out[22]: 'lo o'

In [23]: a[::3] # every three characters, from beginning to end
Out[23]: 'hl r!'
```

Accents and special characters can also be handled in Unicode strings. A string is an immutable object and it is not possible to modify its contents. One may however create new strings from the original one.

```
>>>a = "hello, world!"
>>>a[2] = 'z'
>>>a.replace('l', 'z', 1)
>>>a.replace('l', 'z')
```

```
In [28]: a = "hello, world!"

In [29]: print(a)
hello, world!

In [30]: a[2] = 'z'
Traceback (most recent call last):

  File "<ipython-input-30-1252fe4739cb>", line 1, in <module>
    a[2] = 'z'

TypeError: 'str' object does not support item assignment


In [31]: a.replace('l', 'z', 1)
Out[31]: 'hezlo, world!'

In [32]: a.replace('l', 'z')
Out[32]: 'hezzo, worzd!'
```

Strings have many useful methods, such as a.replace as seen above. Remember the a.object-oriented notation and use tab completion or help(str) to search for new methods. See also: Python offers advanced possibilities for manipulating strings, looking for patterns or formatting. The interested reader is referred to https://docs.python.org/library/stdtypes.html#stringmethodsand https://docs.python.org/library/string.html#new-string-formatting

String formatting:

```
>>> 'An integer: %i; a float: %f; another string: %s' % (1,
0.1, 'string')
>>> i = 102
>>> filename = 'processing_of_dataset_%d.txt' % i
```

```
In [35]: 'An integer: %i; a float: %f; another string: %s' % (1, 0.1, 'string')
Out[35]: 'An integer: 1; a float: 0.100000; another string: string'

In [36]: i = 102

In [37]: filename = 'processing_of_dataset_%d.txt' % i

In [38]: print(filename)
processing_of_dataset_102.txt
```

Python List

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type. Declaring a list is , Items separated by commas are enclosed within brackets [ ].

```
>>> colors = ['red', 'blue', 'green', 'black', 'white']
>>> type(colors)
```

```
In [41]: colors = ['red', 'blue', 'green', 'black', 'white']

In [42]: type(colors)
Out[42]: list
```

Indexing: accessing individual objects contained in the list:

```
In [43]: colors[2]
Out[43]: 'green'

In [44]: colors[-1]
Out[44]: 'white'

In [45]: colors[-2]
Out[45]: 'black'
```

Indexing starts at 0 (as in C), not at 1 (as in Fortran or Matlab)!,

Slicing: obtaining sublists of regularly-spaced elements. Note that colors[start:stop] contains the elements with indices i such as start<= i < stop (i ranging from start to stop-1). Therefore, colors[start:stop] has (stop - start) elements. Slicing syntax: colors[start:stop:stride] All slicing parameters are optional.

```
In [47]: colors
Out[47]: ['red', 'blue', 'green', 'black', 'white']

In [48]: colors[2:4]
Out[48]: ['green', 'black']

In [49]: colors[3:]
Out[49]: ['black', 'white']

In [50]: colors[:3]
Out[50]: ['red', 'blue', 'green']

In [51]: colors[::2]
Out[51]: ['red', 'green', 'white']
```

Lists are mutable objects and can be modified:

```
In [53]: colors[0] = 'yellow'

In [54]: colors
Out[54]: ['yellow', 'blue', 'green', 'black', 'white']

In [55]: colors[2:4] = ['gray', 'purple']

In [56]: colors
Out[56]: ['yellow', 'blue', 'gray', 'purple', 'white']
```

Note: The elements of a list may have different types:

```
In [59]: colors = [3, -200, 'hello']

In [60]: colors
Out[60]: [3, -200, 'hello']

In [61]: colors[1], colors[2]
Out[61]: (-200, 'hello')
```

Add and remove elements:

```
In [63]: colors = ['red', 'blue', 'green', 'black', 'white']

In [64]: colors.append('pink')

In [65]: colors
Out[65]: ['red', 'blue', 'green', 'black', 'white', 'pink']

In [66]: colors.pop() # removes and returns the last item
Out[66]: 'pink'

In [67]: colors
Out[67]: ['red', 'blue', 'green', 'black', 'white']

In [68]: colors.extend(['pink', 'purple']) # extend colors, in-place

In [69]: colors
Out[69]: ['red', 'blue', 'green', 'black', 'white', 'pink', 'purple']

In [70]: colors = colors[:-2]

In [71]: colors
Out[71]: ['red', 'blue', 'green', 'black', 'white']
```

Reverse:

```
In [79]: colors = ['red', 'blue', 'green', 'black', 'white']

In [80]: rcolors = colors[::-1]

In [81]: rcolors
Out[81]: ['white', 'black', 'green', 'blue', 'red']

In [82]: rcolors2 = list(colors)

In [83]: rcolors2
Out[83]: ['red', 'blue', 'green', 'black', 'white']

In [84]: rcolors2.reverse() # in-place

In [85]: rcolors2
Out[85]: ['white', 'black', 'green', 'blue', 'red']
```

Concatenate and repeat lists:

```
In [87]: rcolors + colors
Out[87]:
['white',
 'black',
 'green',
 'blue',
 'red',
 'red',
 'blue',
 'green',
 'black',
 'white']

In [88]: rcolors * 2
Out[88]:
['white',
 'black',
 'green',
 'blue',
 'red',
 'white',
 'black',
 'green',
 'blue',
 'red']
```

Sort:

```
In [90]: sorted(rcolors) # new object
Out[90]: ['black', 'blue', 'green', 'red', 'white']

In [91]: rcolors
Out[91]: ['white', 'black', 'green', 'blue', 'red']

In [92]: rcolors.sort() # in-place

In [93]: rcolors
Out[93]: ['black', 'blue', 'green', 'red', 'white']
```

Methods and Object-Oriented Programming

The notation rcolors.method() (e.g. rcolors.append(3) and colors.pop()) is our first example of object- oriented programming (OOP). Being a list, the object *rcolors* owns the *method function* that is called using the notation.

```
>>>rcolors.<tab> # views the list of OOP function that can be performed
on this object.
```

Python Tuple

Tuples are basically immutable lists. The elements of a tuple are written between parentheses, or just separated by commas

```
In [1]: t = 12345, 54321, 'hello!'

In [2]: t[0]
Out[2]: 12345

In [3]: t[0]=321
Traceback (most recent call last):

  File "<ipython-input-3-fd234f9da6c0>", line 1, in <module>
    t[0]=321

TypeError: 'tuple' object does not support item assignment
```

Python Set

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }.

Items in a set are not ordered.

```
In [7]: s = set(('a', 'b', 'c', 'a'))

In [8]: a={10,20,30,40,50}

In [9]: type(s)
Out[9]: set

In [10]: type(a)
Out[10]: set

In [11]: s = {10, 20, 20, 30, 30, 30}
    ...: print(s)                    #automatically set won't consider duplicate elements
{10, 20, 30}

In [12]: print(s[1]) #we can't print particular element in set because
    ...:             #it's unorder collections of items
Traceback (most recent call last):

  File "<ipython-input-12-3e2f312e6983>", line 1, in <module>
    print(s[1]) #we can't print particular element in set because

TypeError: 'set' object is not subscriptable
```

Python Dictionary

Dictionary is an unordered collection of key-value pairs. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

```
In [16]: tel = {'emmanuelle': 5752, 'sebastian': 5578}

In [17]: tel['francis'] = 5915 # add an entry

In [18]: tel
Out[18]: {'emmanuelle': 5752, 'sebastian': 5578, 'francis': 5915}

In [19]: tel['sebastian']
Out[19]: 5578

In [20]: tel.keys()
Out[20]: dict_keys(['emmanuelle', 'sebastian', 'francis'])

In [21]: tel.values()
Out[21]: dict_values([5752, 5578, 5915])

In [22]: 'francis' in tel
Out[22]: True

In [23]: tel['hammer']
Traceback (most recent call last):

  File "<ipython-input-23-a6171a5b2f96>", line 1, in <module>
    tel['hammer']
```

Conversion between Datatypes

We can convert between different data types by using different type conversion functions like int(), float(), str() etc.

```
>>>float(5)      #convert interger to float using float() method
Out[25]: 5.0
>>>int(100.5)    #convert float to integer using int() method
Out[26]: 100
>>>str(20)       #convert integer to string
Out[27]: '20'
>>>user = "Amir"
>>>lines = 100
>>>print("Congratulations, " + user + "! You just wrote " + str(lines) +
" lines of code" )
#remove str and gives error
```

```
 Congratulations, Amir! You just wrote 100 lines of code
 >>>a = [1, 2, 3]
 >>>print(type(a))       #type of a is list
<class 'list'>
 >>>s = set(a)           #convert list to set using set() method
 >>>print(type(s))       #now type of s is set
<class 'set'
>>>list("Hello")        #convert String to list using list() method
Out[30]: ['H', 'e', 'l', 'l', 'o']
```

Python Input and Output

Python Output

We use the print() function to output data to the standard output device

```
In [39]: print("Hello World")
Hello World

In [40]: a = 10

In [41]: print("The value of a is", a)
The value of a is 10

In [42]: a = 10; b = 20 #multiple statements in single line.
    ...:
    ...: print("The value of a is {} and b is {}".format(a, b))    #default
The value of a is 10 and b is 20

In [43]: a = 10; b = 20  #multiple statements in single line
    ...:
    ...: print("The value of b is {1} and a is {0}".format(a, b)) #specify position of
arguments
The value of b is 20 and a is 10

In [44]: #we can use keyword arguments to format the string
    ...: print("Hello {name}, {greeting}".format(name="Amir", greeting="Good Morning"))
Hello Amir, Good Morning

In [45]: #we can combine positional arguments with keyword arguments
    ...: print('The story of {0}, {1}, and {other}'.format('Bill', 'Manfred',
    ...:                                                  other='Georg'))
The story of Bill, Manfred, and Georg
```

Python Input

If we want to take the input from the user, In Python, we have the input() function to allow this

```
In [49]: num = input("Enter a number: ")

Enter a number: 10

In [50]: print num
  File "<ipython-input-50-f59b73ad6832>", line 1
    print num
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(num)?


In [51]: print (num)
10
```

Operators

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

Operator Types

1. Arithmetic operators
2. Comparison (Relational) operators
3. Logical (Boolean) operators
4. Bitwise operators
5. Assignment operators
6. Special operators

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

+ , -, *, /, %, //, ** are arithmetic operators

```
In [61]: x, y = 10, 20

In [62]: print(x + y) # addition
30

In [63]: print(x - y) # subtraction
-10

In [64]: print(x * y) # multiplication
200

In [65]: print(x / y) # division
0.5

In [66]: print(x % y) # #modulo division (%)
10

In [67]: print(x // y) #Floor Division (//)
0

In [68]: print(x ** y) #Exponent (**)
100000000000000000000
```

Comparison Operators

Comparison operators are used to compare values. It either returns True or False according to the condition.

>, <, ==, !=, >=, <= are comparison operators

```
In [74]: a, b = 10, 20

In [75]: print(a < b)  #check a is less than b
True

In [76]: print(a > b) #check a is greater than b
False

In [77]: print(a == b)#check a is equal to b
False

In [78]: print(a != b)#check a is not equal to b (!=)
True

In [79]: print(a >= b)#check a greater than or equal to b
False

In [80]: print(a <= b)#check a less than or equal to b
True
```

Logical Operators

Logical operators are **and, or, not** operators.

```
In [92]: a, b = True, False

In [93]: print(a and b) #print a and b
False

In [94]: print(a or b) #print a or b
True

In [95]: print(not b) #print not b
True
```

Bitwise operators

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit

&, |, ~, ^, >>, << are Bitwise operators

```
In [106]: a, b = 10, 4
     ...: print(a & b) #Bitwise AND
     ...: print(a | b) #Bitwise OR
     ...: print(~ b)#Bitwise NOT
     ...: print(a ^ b)#Bitwise XOR
     ...: print(a >> b)#Bitwise rightshift
     ...: print(a << b)#Bitwise Leftshift
0
14
-5
14
0
160
```

Assignment operators

Assignment operators are used in Python to assign values to variables. a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

```
In [107]: a = 10
     ...:
     ...: a += 10         #add AND
     ...: print(a)
     ...: a -= 10 #subtract AND (-=)
     ...: print(a)
     ...: a *= 10 #Multiply AND (*=)
     ...: print(a)
     ...:
     ...: a /= 10 #Divide AND (/=)
     ...: print(a)
     ...:
     ...: a %= 10  #Modulus AND (%=)
     ...: print(a)
     ...:
     ...: a //= 10  #Floor Division (//=)
     ...: print(a)
     ...:
     ...: a **= 10  #Exponent AND (**=)
     ...: print(a)
20
10
100
10.0
0.0
0.0
0.0
```

Special Operators

Identity Operators

is and is not are the identity operators in Python.

They are used to check if two values (or variables) are located on the same part of the memory.

```
In [110]: a = 5
     ...: b = 5
     ...: print(a is b)    #5 is object created once both a and b points to same object
     ...: print(a is not b)  #check is not
True
False
```

Membership Operators

in and not in are the membership operators in Python.

They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

```
In [113]: lst = [1, 2, 3, 4]

In [114]: print(1 in lst)       #check 1 is present in a given list or not
True

In [115]: print(5 in lst) #check 5 is present in a given list
False

In [116]: d = {1: "a", 2: "b"}

In [117]: print(1 in d)
True
```

Lab Tasks:

1. Write a Python program to sum all the items in a list.
2. Write a Python program to get the largest number from a list.
3. Write a Python program to remove duplicates from a list
4. Write a Python program to convert list to list of dictionaries.
   Sample lists: ["Black", "Red", "Maroon", "Yellow"], ["#000000", "#FF0000", "#800000", "#FFFF00"]
   Expected Output: [{'color_name': 'Black', 'color_code': '#000000'}, {'color_name': 'Red', 'color_code': '#FF0000'}, {'color_name': 'Maroon', 'color_code': '#800000'}, {'color_name': 'Yellow', 'color_code': '#FFFF00'}]
5. Write a Python program to read a matrix from console and print the sum for each column. Accept matrix rows, columns and elements for each column separated with a space(for every row) as input from the user.
   Input rows: 2

Input columns: 2
Input number of elements in a row (1, 2, 3):
1 2
3 4
sum for each column:
4 6

6. Write a Python program to Zip two given lists of lists.
Original lists:
[[1, 3], [5, 7], [9, 11]]
[[2, 4], [6, 8], [10, 12, 14]]
Zipped list:
[[1, 3, 2, 4], [5, 7, 6, 8], [9, 11, 10, 12, 14]]

7. Write a Python program to extract the nth element from a given list of tuples.
Original list:
[('Greyson Fulton', 98, 99), ('Brady Kent', 97, 96), ('Wyatt Knott', 91, 94), ('Beau Turnbull', 94, 98)]
Extract nth element (n = 0) from the said list of tuples:
['Greyson Fulton', 'Brady Kent', 'Wyatt Knott', 'Beau Turnbull']
Extract nth element (n = 2) from the said list of tuples:
[99, 96, 94, 98]

8. Write a Python program to remove additional spaces in a given list.
Original list:
['abc ', ' ', ' ', 'sdfds ', ' ', ' ', 'sdfds ', 'huy']
Remove additional spaces from the said list:
['abc', '', '', 'sdfds', '', '', 'sdfds', 'huy']

9. Write a Python program to multiply all the items in a dictionary

10. Write a Python program to print all unique values in a dictionary.
Sample Data : [{"V":"S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"}, {"VII":"S005"}, {"V":"S009"},{"VIII":"S007"}]
Expected Output : Unique Values: {'S005', 'S002', 'S007', 'S001', 'S009'}

11. Write a Python program to create a dictionary of keys x, y, and z where each key has as value a list from 11-20, 21-30, and 31-40 respectively. Access the fifth value of each key from the dictionary.
{'x': [11, 12, 13, 14, 15, 16, 17, 18, 19],
'y': [21, 22, 23, 24, 25, 26, 27, 28, 29],
'z': [31, 32, 33, 34, 35, 36, 37, 38, 39]}
15
25
35
x has value [11, 12, 13, 14, 15, 16, 17, 18, 19]
y has value [21, 22, 23, 24, 25, 26, 27, 28, 29]
z has value [31, 32, 33, 34, 35, 36, 37, 38, 39]

12. Write a Python program to print a tuple with string formatting.

Sample tuple : (100, 200, 300)

Output : This is a tuple (100, 200, 300)

13. Write a Python program to replace last value of tuples in a list.

Sample list: [(10, 20, 40), (40, 50, 60), (70, 80, 90)]

Expected Output: [(10, 20, 100), (40, 50, 100), (70, 80, 100)]

14. Write a Python program to find the elements in a given set that are not in another set.

15. Write a Python program to check a given set has no elements in common with other given set.

16. Write a Python program that accept name of given subject and marks. Input number of subjects in first line and subject name,marks separated by a space in next line. Print subject name and marks in order of its first occurrence.

Sample Output:
Powered by
Number of subjects: 3
Input Subject name and marks: Urdu 58
Input Subject name and marks: English 62
Input Subject name and marks: Math 68
Urdu 58
English 62
Math 68