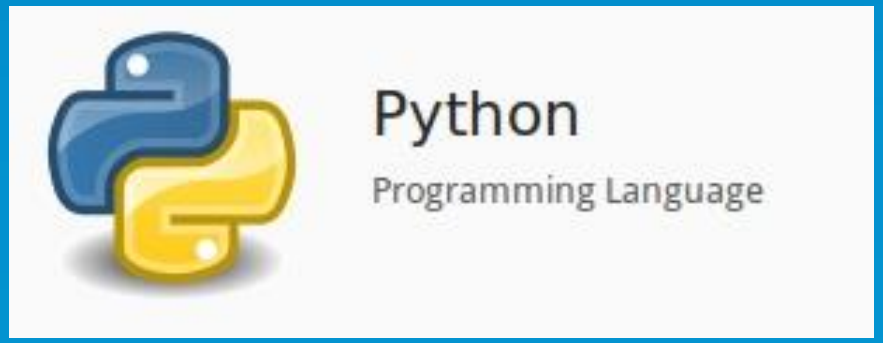


Python Programming

STRINGS

2019

String Handling

WHAT IS STRING

What is string?

- A string is a sequence of characters. A character is simply a symbol. For example, the English language has 26 characters.
- Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0's and 1's.
- This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encoding used.
- In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding.

What is string?

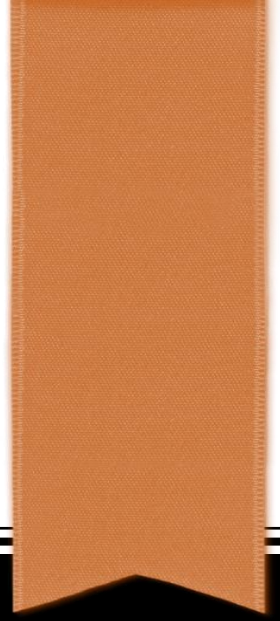
- Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable.
- For example –

  var1 = 'Hello World!'

  var2 = "Python Programming"

  var3 = '''This is also valid'''

  var4 = """So is this"""



STRING OPERATIONS AND INDICES

String operations and indices

- Python does not support a character type; these are treated as strings of length one, thus also considered a substring.
- To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. For example –

```
#!/usr/bin/python
var1 = 'Hello World!'
var2 = "Python Programming"
print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

- When the above code is executed, it produces the following result

```
var1[0]:  H
var2[1:5]:  ytho
```

String operations and indices

- If we try to access index out of the range or use decimal number, we will get errors.

```
# index must be in range
>>> my_string[15]
...
IndexError: string index out of range
# index must be an integer
>>> my_string[1.5]
...
```

- TypeError: string indices must be integers

String operations and indices

- We can access individual characters using indexing and a range of characters using slicing. Index starts from 0.
- Trying to access a character out of index range will raise an IndexError. The index must be an integer.
- We can't use float or other types, this will result into TypeError.
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.
- We can access a range of items in a string by using the slicing operator (colon).

String operations and indices

- Slicing can be best visualized by considering the index to be between the elements as shown below.
- If we want to access a range, we need the index that will slice the portion from the string.

e	K	A	L	A	S	A	A	L	A
0	1	2	3	4	5	6	7	8	9
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

String operations and indices

```
str = 'programiz'
print('str = ', str)
#first character
print('str[0] = ', str[0])
#last character
print('str[-1] = ', str[-1])
#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])
#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

BASIC STRING OPERATIONS

Basic String Operations

- Delete a string
- Operation 1: How to change or delete a string?
- Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. We can simply reassign different strings to the same name.

```
>>> my_string = ‘eKalasaala'
>>> my_string[5] = ‘Z’
...
TypeError: 'str' object does not support item assignment
>>> my_string = 'Python'
>>> my_string
'Python'
```

Basic String Operations

- Delete a string
- We cannot delete or remove characters from a string. But deleting the string entirely is possible using the keyword del.

```
>>> del my_string[1]
...
TypeError: 'str' object doesn't support item deletion
>>> del my_string
>>> my_string
...
NameError: name 'my_string' is not defined
```

- There are many operations that can be performed with string which makes it one of the most used datatypes in Python

Basic String Operations

- String Multiplication and Concatenation
- Concatenation of Two or More Strings
  - Joining of two or more strings into a single one is called concatenation.
  - The + operator does this in Python. Simply writing two string literals together also concatenates them.
  - The * operator can be used to repeat the string for a given number of times.

```
str1 = 'Hello'
str2 ='World!'
# using +
print('str1 + str2 = ', str1 + str2)
# using *
print('str1 * 3 =', str1 * 3)
```

Basic String Operations

- String Multiplication and Concatenation
- Concatenation of Two or More Strings
  - Writing two string literals together also concatenates them like + operator.
  - If we want to concatenate strings in different lines, we can use parentheses.

```
>>> # two string literals together
>>> 'Hello ''World!'
'Hello World!'
>>> # using parentheses
>>> s = ('Hello '
...      'World')
>>> s
'Hello World'
```

Basic String Operations

- Iterating Through String
- Using for loop we can iterate through a string. Here is an example to count the number of 'l' in a string.

```
count = 0
for letter in 'Hello World':
    if(letter == 'l'):
        count += 1
print(count,'letters found')
```

Basic String Operations

- String Membership Test
- We can test if a sub string exists within a string or not, using the keyword in.

```
>>> 'a' in 'program'
True
>>> 'at' not in 'battle'
False
```

Basic String Operations

- Built-in functions to Work with Python
  - Various built-in functions that work with sequence, works with string as well.
  - Some of the commonly used ones are enumerate() and len().
  - The enumerate() function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.
  - Similarly, len() returns the length (number of characters) of the string.

```
str = 'cold'
# enumerate()
list_enumerate = list(enumerate(str))
print('list(enumerate(str) = ', list_enumerate)
#character count
print('len(str) = ', len(str))
```

- 9 ->

STRING FORMATTING OPERATOR

String Formatting Operator

- Python String Formatting
- Escape Sequence
- If we want to print a text like -He said, "What's there?"- we can neither use single quote or double quotes. This will result into Syntax Error as the text itself contains both single and double quotes.

```
>>> print("He said, "What's there?"")
...
SyntaxError: invalid syntax
>>> print('He said, "What's there?"')
...
SyntaxError: invalid syntax
```

String Formatting Operator

- Python String Formatting

- One way to get around this problem is to use triple quotes. Alternatively, we can use escape sequences.
- An escape sequence starts with a backslash and is interpreted differently. If we use single quote to represent a string, all the single quotes inside the string must be escaped. Similar is the case with double quotes. Here is how it can be done to represent the above text.

```
# using triple quotes
print('''He said, "What's there?"''')
# escaping single quotes
print('He said, "What\'s there?"')
# escaping double quotes
print("He said, \"What's there?\"")
```

String Formatting Operator

- Python String Formatting

Escape Sequence in Python

Escape Sequence	Description
\newline	Backslash and newline ignored
\\	Backslash
\'	Single quote
\"	Double quote
\a	ASCII Bell
\b	ASCII Backspace
\f	ASCII Formfeed
\n	ASCII Linefeed
\r	ASCII Carriage Return
\t	ASCII Horizontal Tab
\v	ASCII Vertical Tab
\ooo	Character with octal value ooo
\xHH	Character with hexadecimal value HH

String Formatting Operator

- Python String Formatting
- Examples:

```
>>> print("C:\\Python32\\Lib")
C:\Python32\Lib
>>> print("This is printed\n in two lines")
This is printed
in two lines
>>> print("This is \x48\x45\x58 representation")
This is HEX representation
```

String Formatting Operator

- Raw String to ignore escape sequence
  - Sometimes we may wish to ignore the escape sequences inside a string. To do this we can place r or R in front of the string. This will imply that it is a raw string and any escape sequence inside it will be ignored.

```
>>> print("This is \x61 \ngood example")
This is a
good example
>>> print(r"This is \x61 \ngood example")
This is \x61 \ngood example
```

String Formatting Operator

- The format() Method for Formatting Strings

- The format() method that is available with the string object is very versatile and powerful in formatting strings.
- Format strings contains curly braces {} as placeholders or replacement fields which gets replaced.
- We can use positional arguments or keyword arguments to specify the order.

String Formatting Operator

- The format() Method for Formatting Strings
- # default(implicit) order

```
default_order = "{}, {} and {}".format('John','Bill','Sean')
print('\n--- Default Order ---')
print(default_order)
# order using positional argument
positional_order = "{1}, {0} and {2}".format('John','Bill','Sean')
print('\n--- Positional Order ---')
print(positional_order)
# order using keyword argument
keyword_order = "{s}, {b} and {j}".format(j='John',b='Bill',s='Sean')
print('\n--- Keyword Order ---')
print(keyword_order)
```

String Formatting Operator

- The format() Method for Formatting Strings

- The format() method can have optional format specifications.
- They are separated from field name using colon.
- For example, we can left-justify <, right-justify > or center ^ a string in the given space.
- We can also format integers as binary, hexadecimal etc. and floats can be rounded or displayed in the exponent format.
- There are a ton of formatting you can use. Visit here for all the string formatting available with the format() method.

String Formatting Operator

- The format() Method for Formatting Strings

```
>>> # formatting integers
>>> "Binary representation of {0} is {0:b}".format(12)
'Binary representation of 12 is 1100'
>>> # formatting floats
>>> "Exponent representation: {0:e}".format(1566.345)
'Exponent representation: 1.566345e+03'
>>> # round off
>>> "One third is: {0:.3f}".format(1/3)
'One third is: 0.333'
>>> # string alignment
>>> "|{:<10}|{:^10}|{:>10}|".format('butter','bread','ham')
'|butter    |  bread   |       ham|'
```

String Formatting Operator

- Old style formatting
  - We can even format strings like the old sprintf() style used in C programming language.
  - We use the % operator to accomplish this.

```
>>> x = 12.3456789
>>> print('The value of x is %3.2f' %x)
The value of x is 12.35
>>> print('The value of x is %3.4f' %x)
The value of x is 12.3457
```

BUILT-IN STRING METHODS

Built-in String Methods

- There are numerous methods available with the string object:

```
>>> “Welcome2eKalasaala".lower()
' welcome2ekalasaala ‘
>>> " Welcome2eKalasaala".upper()
WELCOME2EKALASAALA'
>>> "This will split all words into a list".split()
['This', 'will', 'split', 'all', 'words', 'into', 'a', 'list']
>>> ' '.join(['This', 'will', 'join', 'all', 'words', 'into', 'a', 'string'])
'This will join all words into a string'
>>> 'Happy New Year'.find('ear')
11
>>> 'Happy New Year'.replace('Happy',‘Awesome')
' Awesome New Year'
```

CHAPTER 10 CONTENT TO COME HERE

MORE FUNCTIONS

Built-in String Methods

- A string in Python can be tested for truth value. The return type will be in Boolean value (True or False):
- my_string="Hello World"
- print my_string.isalnum() #False
- print my_string.isalpha() #False
- print my_string.isdigit() #False
- print my_string.istitle() #True
- print my_string.isupper() #False
- print my_string.islower() #False
- print my_string.isspace() #False
- print my_string.endswith('d') #True
- print my_string.startswith('H') #True

PROGRAMS TO PRACTICE

String Program 1

- Write a Python program to move spaces to the front of a given string.

- Input: Hello from Hyderabad
- Output: HellofromHyderabad

Solution Program 1

- inputStr = input("Enter the String: ")
- noSpacesChars = [ch for ch in inputStr if ch!=' ']
- print(noSpacesChars)
- spacesCount = len(inputStr) - len(noSpacesChars)
- result = ' '*spacesCount
- result = '"'+result + ''.join(noSpacesChars)+'"'
- print(result)

String Program 2

- Write a Python program to compute sum of digits of a given string.

- Input: ABC369abc810xyz
- Output:27

Solution Program 2

```
inputStr = input("Enter the String: ")
sumDigit = 0
for ch in inputStr:
    if ch.isdigit() == True:
        chr2int = int(ch)
        sumDigit = sumDigit + chr2int
print("Sum of Digits in the String are: ", sumDigit)
```

String Program 3

- Write a Python program to capitalize first and last letters of each word of a given string.

- Input: hello from hyderabad
- Output: HellO FroM HyderabaD

Solution Program 3

```
inputStr = input("Enter the String: ")
inputStr = inputStr.title()  #Set to Title Case
result = ""
for word in inputStr.split():
    result += word[:-1] + word[-1].upper() + " "
print(result[:-1])
```

String Program 4

- Write a Python program to print the index of the character in a string.

- Input: Hello
- Output:
- Current character h position at 0
- Current character e position at 1
- Current character l position at 2
- Current character l position at 3
- Current character 0 position at 4

Solution Program 4

- inputStr = input("Enter the String: ")
- for counter, ch in enumerate(inputStr):
- print("Current character", ch, "position at", counter)

String Program 5

- Write a Python program to swap comma and dot in a string.

- Input: 75.99,25.11
- Output: 75,99.25,11

Solution Program 5

- amount = "75.99,25.11"
- amountTrans = amount.maketrans('.,', ',.')
- # maketrans() method returns a translation table
- # translate
- amount = amount.translate(amountTrans)
- print(amount)

Programs to Practice

- Swap commas and dots in a String
- Program to convert String to a List
- Count and display vowels in a string
- Python program to check the validity of a Password (Atleast 1 caps, 1 digit and minimum 8 characters)
- Python program to count number of vowels using sets in given string
- Check for URL in a String
- Check if a Substring is Present in a Given String
- Check if two strings are anagram or not (contains same set of characters)
- Map function and Dictionary in Python to sum ASCII values
- Map function and Lambda expression in Python to replace characters
- SequenceMatcher in Python for Longest Common Substring
- Print the initials of a name with last name in full
- Find the k most frequent words from data set in Python
- Find all close matches of input string from a list
- Check if there are K consecutive 1's in a binary number
- Check if both halves of the string have same set of characters in Python
- Find the first repeated word in a string in Python
- Second most repeated word in a sequence in Python
- K'th Non-repeating Character in Python
- Reverse words in a given String in Python
- Print number with commas as 1000 separators in Python
- Remove all duplicates words from a given sentence
- Sort words of sentence in ascending order
- Reverse each word in a sentence
- Python code to print common characters of two Strings in alphabetical order
- Python program to count upper and lower case characters without using inbuilt functions

Thank You

Python Programming Course

Hand Notes 7

Hand Notes for:

Day 13 (May 23, 2020) – Collections (List, Tuples)

Day 14 (May 25, 2020) – Collections(List Functions)

Day 13: List, Tuple

A data structure is a particular way of organizing data in a computer so that it can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. In this chapter we will look at the data structures available in Python programming.

LIST

A list is a data structure that holds an ordered collection of items i.e. you can store a sequence of items in a list. This is easy to imagine if you can think of a shopping list where you have a list of items to buy, except that you probably have each item on a separate line in your shopping list whereas in Python you put commas in between them. The list of items should be enclosed in square brackets so that Python understands that you are specifying a list. Once you have created a list, you can add, remove or search for items in the list. Since we can add and remove items, we say that a list is a mutable data type i.e. this type can be altered.

Properties of List

- Ordered collection of data
- Data can be of different types
- Lists are mutable
- Issues with shared references and mutability
- Same subset operations as Strings

INBUILT FUCTIONS

Method	Description	Example
append()	Adds an element at the end of the list	thislist.append("orange")
clear()	Removes all the elements from the list	thislist.clear()
copy()	Returns a copy of the list	x = fruits.copy()
count()	Returns the number of elements with the specified value	x = fruits.count("cherry")
extend()	Add the elements of a list (or any iterable), to the end of the current list	fruits.extend(cars)
index()	Returns the index of the first element with the specified value	x = fruits.index("cherry")
insert()	Adds an element at the specified position	fruits.insert(1, "orange")
pop()	Removes the element at the specified position	fruits.pop(1)
remove()	Removes the item with the specified value	fruits.remove("banana")
reverse()	Reverses the order of the list	fruits.reverse()
sort()	Sorts the list	fruits.sort()

BASIC LIST OPERATORS

Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership

for x in [1, 2, 3]: print x,	1 2 3	Iteration

TUPLE

Tuples are used to hold together multiple objects. Think of them as similar to lists, but without the extensive functionality that the list class gives you. One major feature of tuples is that they are immutable like strings i.e. you cannot modify tuples. Tuples are defined by specifying items separated by commas within an optional pair of parentheses. Tuples are usually used in cases where a statement or a user-defined function can safely assume that the collection of values i.e. the tuple of values used will not change.

Programs in the Class

```
#Collection - it can store more than 1 value
str1="Hello"  #Strings are immutable
# List
var1 = [5,10,15,20]
print("Values are: ",var1)
print("Type of var1 is ",type(var1))
print(var1[-3:])
var1[1]= 50    #Lists are mutable
print("var1 after second member change is ",var1)

sum=0
for x in var1:
    sum+= x
print("Sum of the values are ",sum)

#marksList = [0]*5
marksList = ['0',0,0.0,False,0]
sum=0
print("Length of the list is ",len(marksList))
for i in range(0,5):
    marksList[i] = int(input("Enter the marks in Subject "+str(i+1)+" : "))
    sum+=marksList[i]
print("You have entered following marks: ",marksList, "and total marks is ",sum)

marksList2=[]
sum=0
print("Length of the list is ",len(marksList))
for i in range(0,5):
    var1 = int(input("Enter the marks in Subject "+str(i+1)+" : "))
    marksList2.append(var1)
    sum+=marksList2[i]
    #sum+=var1
print("You have entered following marks: ",marksList2, "and total marks is ",sum)
print("Now the length is ",len(marksList2))
```

```
# Inbuilt functions of List
fruits1 =[]  #create list
fruits1.insert(1,"Orange")
```

```
fruits1.insert(0,"Mango")
fruits1.insert(2,"Banana")
print("Fruits list has ",fruits1)
fruits1.pop(1)
print("Fruits list after POP(1) ",fruits1)
fruits1.remove("Banana")
print("Fruits list after Remove(Banana) ",fruits1)
fruits1.insert(1,"Orange")
fruits1.insert(2,"Banana")
fruits1.insert(3,"Cherry")
fruits1.insert(4,"Apple")

fruit2check = "Pineapple"
print(fruit2check in fruits1)
fruits1.sort()
print("Sorted order is ", fruits1)
```

```
###Tuple

var1=(1,2,3)
print("Values in Tuple ",var1)
print("Second Value in Tuple ",var1[1])
#var1[1] = 5   #Immutable
fruits1 = []
fruits1.insert(1,"Orange")
fruits1.insert(0,"Mango")
fruits1.insert(2,"Banana")
print("Type of Fruits ",type(fruits1))
fruits1 = tuple(fruits1)
print("Type of Fruits ",type(fruits1))
fruits1 = list(fruits1)
print("Type of Fruits ",type(fruits1))

#Tuples are very similar to List but doesnt allow mutability because of which
# it is faster to do programs using Tuples
```

More on Tuples

A tuple is just like a list of a sequence of immutable python objects.

The difference between list and tuple is that list are declared in square brackets and can be changed while tuple is declared in parentheses and cannot be changed.

However, you can take portions of existing tuples to make new tuples.

Syntax

Tup = ('Jan','feb','march')

Tuples Programs in the Class

```
 #Comparing Tuples
#A comparison operator in Python can work with tuples.
#The comparison starts with a first element of each tuple.
#If they do not compare to =,< or > then it proceed to the second element and so on.
#It starts with comparing the first element from each of the tuples.
```

```
#case 1
a=(5,6)
b=(1,4)
if (a>b):print("a is bigger")
else: print("b is bigger")
#case 2
a=(5,6)
b=(5,4)
if (a>b):print("a is bigger")
else: print ("b is bigger")
#case 3
a=(5,6)
b=(6,4)
if (a>b):print("a is bigger")
else: print("b is bigger")
```

```
#Packing and Unpacking
#In packing, we place value into a new tuple while in unpacking we extract those values
back into variables.
x = ("Quest Learning", 10, "Education")     # tuple packing
(company, emp, profile) = x     # tuple unpacking
print(company)
print(emp)
print(profile)
```

```
#Creating Nested Tuple
#Using a for loop we can iterate though each item in a tuple.
for name in ('Sachin','Sourav','Rahul','Sehwag'):
          print("Hello",name)
```

```
#Using Tuples  as keys in Dictionaries
#Since tuples are hashable, and list is not, we must use tuple as the key if we need to
create a composite key to use in a dictionary.
#Example
#We would come across a composite key if we need to create a telephone directory that
maps, first-name, last-name, pairs of telephone numbers, etc:
#   directory[last,first] = number
#Implementation:
directory = {}
directory[("Sachin","Tendulkar")] = 33338888
directory[("Mahi","Dhoni")] = 44448888

#Inside the brackets, the expression is a tuple. We could use tuple assignment in a for
loop to navigate this dictionary.
for last, first in directory:
   print(first, last, directory[last, first])
#This loop navigates the keys in the directory, which are tuples. It assigns the
elements of each tuple to last and first and then prints the name and corresponding
telephone number.
```

```
#Deleting Tuples
#Tuples are immutable and cannot be deleted, but
# deleting tuple entirely is possible by using the keyword "del."
del a
```

```
#Slicing of Tuple
```

```
#To fetch specific sets of sub-elements from tuple or list, we use this unique function called slicing. Slicing is not only applicable to tuple but also for array and list.
x = ("a", "b","c", "d", "e")
print(x [2:4])
```

```
#Tuple Membership Test
#We can test if an item exists in a tuple or not, using the keyword in.
my_tuple = ('a','p','p','l','e',)
print('a' in my_tuple)
#  Output: True
print('b' in my_tuple)
#  Output: False
# Not in operation
print('g' not in my_tuple)
#  Output: True
```

Day 14: List Functions & Dictionary

LAMBDA

lambda operator or lambda function is used for creating small, one-time and anonymous function objects in Python.
Basic Syntax: lambda arguments : expression
lambda operator can have any number of arguments, but it can have only one expression. It cannot contain any statements and it returns a function object which can be assigned to any variable.

LAMBDA Programs in the Class

```
#Normal Way of Writing Function:
def power(x, y):
    return x ** y

# Call the function
a =power(2, 3)
print(a)
```

```
#Function using Lambda Operator:
add = lambda x, y: x ** y
a = add(2, 3)
print(a)
```

LIST FUNCTIONS

These are three functions which facilitate a functional approach to programming. We will discuss them one by one and understand their use cases:

- Map
- Filter
- Reduce

Map

- We spend lot of time analyzing list of data like: List of stock prices, Orders, Customer profiles
- Map applies a function to all the items in an input_list
- map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)

Problem Blueprint:

Data: a1, a2, … ,an
Function: f
map(f, data) returns Iterator over: f(a1), f(a2), …, f(an)

Blueprint

map(function_to_apply, list_of_inputs)

Most of the times we want to pass all the list elements to a function one-by-one and then collect the output.

MAPS Programs in the Class

```
#Program without Maps
items = [1, 2, 3, 4, 5]
squared = []
for i in items:
    squared.append(i**2)
    print(squared)

#Map allows us to implement this in a much simpler and nicer way. Here you go
items = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, items))
```

```
def multiply(x):
    return (x * x)

def add(x):
    return (x + x)

funcs = [multiply, add]
for i in range(5):
    value = list(map(lambda x: x(i), funcs))
    print(value)
```

FILTER

As the name suggests, filter creates a list of elements for which a function returns true. It filters out the data you do not need. E.g. Find all data above average

The filter resembles a for loop but it is a builtin function and faster.

Here is a short and concise example:

FILTER Programs in the Class

```
number_list = range(-5, 5)
less_than_zero = list(filter(lambda x: x < 0, number_list))
```

```
print(less_than_zero)
```

```
import statistics
data_list = [5, 1, 8,2,5,9,12,15,18]
avg_val = statistics.mean (data_list)
above_avg = list(filter(lambda x: x > avg_val, data_list))
print(above_avg)

#Another example, to filter out missing data
states = ["","Telangana", "","Bihar", "","Goa", "","","","Rajasthan", "",""]
final_list = list(filter(None, states))
print(final_list)
```

Reduce

- Reduce is a really useful function for performing some computation on a list and returning the result.
- It applies a rolling computation to sequential pairs of values in a list.
- For example, if you wanted to compute the product of a list of integers.
- Normal way you might go about doing this task in python is using a basic for loop
- In Python 3, it is no longer a built-in function, we need to call from functools.reduce()

How it works

Data: [a1, a2, a3, …, an]
Function: f(x,y)
reduce (f,data):
    Step 1: val1 = f(a1,a2)
    Step 2: val2 = f(val1,a3)
    Step 3: val3 = f(val2,a4)
    …
    Step n-1: valn-1 = f(valn-2,an)
    Return Valn-1

i.e. Reduce: f (f (f (a1,a2), a3), … , an)

REDUCE Programs in the Class

```
# Without Reduce
product = 1
list = [1, 2, 3, 4]
for num in list:
    product = product * num
print("Without Reduce concept, output: ", product)
# product = 24
#Now let's try it with reduce: same logic
from functools import reduce
product = reduce((lambda x, y: x * y), [1, 2, 3, 4])
print("Now with Reduce, output: ", product)
```

Assignments

1. Write a Python program to sum all the items in a list.
2. Write a Python program to multiplies all the items in a list.
3. Write a Python program to get the largest number from a list.
4. Write a Python program to get the smallest number from a list.
5. Write a Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings.

   Sample List : ['abc', 'xyz', 'aba', '1221']

   Expected Result : 2

6. Write a Python program to get a list, sorted in increasing order by the last element in each tuple from a given list of non-empty tuples.

   Sample List : [(2, 5), (1, 2), (4, 4), (2, 3), (2, 1)]

   Expected Result : [(2, 1), (1, 2), (2, 3), (4, 4), (2, 5)]

7. Write a Python program to remove duplicates from a list.
8. Write a Python program to check a list is empty or not.
9. Write a Python program to clone or copy a list.
10. Write a Python program to find the list of words that are longer than n from a given list of words.
11. Write a Python function that takes two lists and returns True if they have at least one common member.
12. Write a Python program to print a specified list after removing the 0th, 4th and 5th elements.

    Sample List : ['Red', 'Green', 'White', 'Black', 'Pink', 'Yellow']

    Expected Output : ['Green', 'White', 'Black']

13. Write a Python program to generate a 3*4*6 3D array whose each element is *.
14. Write a Python program to print the numbers of a specified list after removing even numbers from it.
15. Write a Python program to shuffle and print a specified list.
16. Write a Python program to generate and print a list of first and last 5 elements where the values are square of numbers between 1 and 30 (both included).
17. Write a Python program to generate and print a list except for the first 5 elements, where the values are square of numbers between 1 and 30 (both included).
18. Write a Python program to generate all permutations of a list in Python.
19. Write a Python program to get the difference between the two lists.
20. Write a Python program access the index of a list.

- - - End of Session 4 - - -

Python Programming Course

Hand Notes 8

Hand Notes for:

Day 15 (May 26, 2020) – (Collection - Dictionary)

DICTIONARY

A dictionary is like an address-book where you can find the address or contact details of a person by knowing only his/her name i.e. we associate keys (name) with values (details). A dictionary is an associative array (also known as hashes). Any key of the dictionary is associated (or mapped) to a value. The values of a dictionary can be any Python data type. So dictionaries are unordered key-value-pairs. Below example is a simple German-English dictionary:

```
# -*- coding: iso-8859-1 -*-
en_de = {"red" : "rot", "green" : "grün", "blue" : "blau", "yellow":"gelb"}
print en_de
print en_de.items()
print en_de.keys()
print "Translating 'red' to German: " , en_de["red"]
```

Output:

```
{'blue': 'blau', 'green': 'gr\xfcn', 'yellow': 'gelb', 'red': 'rot'}
[('blue', 'blau'), ('green', 'gr\xfcn'), ('yellow', 'gelb'), ('red', 'rot')]
['blue', 'green', 'yellow', 'red']
Translating 'red' to German:  rot
```

- Python Dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.
- Dictionaries are optimized to retrieve values when the key is known.
- Creating a dictionary is as simple as placing items inside curly braces { } separated by comma.
- An item has a key and the corresponding value expressed as a pair, key: value.
- While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

Method	Description	Example
clear()	Remove all items form the dictionary.	thisdict.clear()
copy()	Return a shallow copy of the dictionary.	dict2 = thisdict.copy()
fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v(defaults to None).	thisdict = dict.fromkeys(x, y)
get(key[,d])	Return the value of key. If key doesn't exit, return d (defaults to None).	x = car.get("model")
items()	Return a new view of the dictionary's items (key, value).	x = thisdict.items()
keys()	Return a new view of the dictionary's keys.	x = thisdict.keys()
pop(key[,d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.	thisdict.pop("model")
popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.	thisdict.popitem()
setdefault(key[,d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).	x = thisdict.setdefault("model", "Brand")
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.	thisdict.update({"color": "White"})

values()	Return a new view of the dictionary's values	x = thisdict.values()

Programs in the Class

```
# empty dictionary
my_dict = {}
# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}
print("1. Dictionary: ", my_dict)
# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}
print("2. Dictionary: ", my_dict)
# using dict()
### As you can see below, we can also create a dictionary using the built-in function dict().
my_dict = dict({1:'apple', 2:'ball'})
print("3. Dictionary: ", my_dict)
# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
print("4. Dictionary: ", my_dict)
```

```
marks = {}.fromkeys(['Math','English','Science'], 0)
for item in marks.items():
      print(item)
#Return Value from copy()
original = {1:'one', 2:'two'}
new = original.copy()
print('Orignal: ', original)
print('New: ', new)
new.clear()
print("New:" , new)
print("Original:", original)

#The method update() adds dictionary dict2's key-values pairs in to dict.
# This function does not return anything
dict = {1:'one', 2:'two'}
dict2 = {3:'three', 4:'four'}
dict.update(dict2)
print("dict is now: ",dict)
```

```
#Delete Keys from the Dictionary
dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
del dict['Charlie']
print("dict : ",dict)

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
for key in Dict.keys():
    if key in Boys.keys():
        print(True)
    else:
        print(False)
```

```
#Run below list example and see the output
```

```
#Did you expect the same output?
list1 = [3,4,5]
list2=list1
list3=list1.copy()    #Shallow copy
list1.append(6)
print("List1 values: ",list1)
print("List2 values: ",list2)
print("List3 values: ",list3)
```

```
list1=[2,4,6]
Dictlist1={0:2,1:4,2:6}
tup1=(1,3,5)

print("Tuple: ",type(tup1))
print(tup1[1])

dict1={"key":"value",46:"Sachin tendulkar", 34:["Rahul Dravid",10000,231]}  #{} for
Dictionary
print(dict1["key"])
print(dict1[46])
print(dict1[34])

#FromKeys - will assign same values to multiple keys
dict2={}.fromkeys(["Science","Maths","Social"],0)
print(dict2)

#Iterate through the dictionary: items is combination of Key & Value
for item in dict2.items():
      print(item)
#Sort dictionary based on keys
print(list(sorted(dict2.keys())))

#Update function of Dictionary
dict1.update(dict2)
print("Dict 1: ", dict1)
print("Dict 2: ", dict2)

#
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
for key in Dict.keys():
    if key in Boys.keys():
        print(True)
    else:
        print(False)


#======== Copy in Lists ================
list1 = [2,4,6,8]
list2=list1  # Deep Copy
list3=list1.copy()  #Shallow Copy
list1.append(10)
list2.append(12)
print("List 1: ", list1)  #2,4,6,8,10
print("List 2: ", list2)  #2,4,6,8,10  |8
print("List 3: ", list3)  #2,4,6,8,10  | 8
```

Assignments

1. # WAP to create a dictionary to use roll no. as Key and Value to be a list with Student Name, class, favorite subject. Write this using input and loops to create this dictionary and perform all the functions discussed in the class

- - - End of Session 8 - - -