PMDS508L - Python Programming
Data Collections

Dr. B.S.R.V. Prasad
Department of Mathematics
School of Advanced Sciences
Vellore Institute of Technology
Vellore

srvprasad.bh@gmail.com (Personal)
srvprasad.bh@vit.ac.in (Official)
+91-8220417476

Python Strings

- String literals in Python are surrounded by either single quotes or double quotes.
- Example: `a = 'Hello'` is same as `a = "Hello"`
- Multiline Strings:

```
a = """This is a long string,
This is the second line of the string,
This is the last line of the string."""
print(a)
```

- We can also use three single quotes instead of three double quotes.
- Note that, the line breaks are inserted at the same position as in the code.

Strings as Arrays

- Strings in Python are represented as arrays of bytes of unicode characters.
- In Python we don't have a character data type, a single character is simply as string of length 1.
- Square brackets can be used to access elements of the string.

```
1 s = "Hello World!"
2 print(s[1])
3 print(s[3])
```

Please note that in Python the index starts with 0 not with 1

- The Length of the string can be found by using `len()` command. `len(s)` gives the length of the string `s`.

String Slicing

- We can return a range of characters by using the slice syntax.
- For slicing the string and return the part of the string, we need to specify the start index and end index, separated by a colon.
- The end index character will not be included in the sliced string i.e., the string returned will begin from the start index character and till up to the end-1 character.

```
1 s = 'Hello World!'
2 print(s[2:5])
```

Negative Indexing in Strings

- We can use the negative indexes to start the slice from the end of the string
- To get the characters from 5 to position 1 (not included) counted from backwards, we cause the following command:

```
s = 'Hello World!'
print(s[-5:-2])
```

String Concatenation

- To concatenate, or combine, strings we can use + operator.

```
a = "Hello"
b = "World!"
c = a+b
print(c) #Prints "HelloWorld!"
c = a + " " + b
print(c) #Prints "Hello World!"
```

String Format

- Please recall that, we cannot combine strings and numbers directly in Python i.e., the code:

```
age = 36
name = "Tom"
txt = name + " age is: " + age
print(txt)
```

returns an error.

- One way to overcome is to convert the integer into string:

```
age = 36
name = "Tom"
txt = name + " age is: " + str(age)
print(txt)
```

String Format

- Another way to combine strings and number is by using `format()` method.
- The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders `{}` are:

```
age = 36
txt = "Tom age is: {}"
print(txt.format(age))
```

String Format

- The `format()` method takes unlimited number of arguments, and are placed into the respective placeholders:

```
name = "Tom"
age = 36
txt = "{} age is: {}"
print(txt.format(name,age))
```

- We can use index numbers `{0}` to be sure the arguments are placed in the correct placeholders:

```
age = 36
name = "Tom"
txt = "{1} age is: {0}"
print(txt.format(age,name))
```

String Format


```
1 quantity = 3
2 itemno = 567
3 price = 4000.60
4 myorder = "We need to by {2} items each of quantity {1}
      and I have a total of {0} rupees with me."
5 print(myorder.format(price, quantity, itemno))
```

f-strings in Python

- Python offers a powerful feature called f-strings (formatted string literals) to simplify string formatting and interpolation.
- `f-strings` is introduced in Python 3.6.
- `f-strings` provide a concise and intuitive way to embed expressions and variables directly into strings.
- To create an `f-string`, prefix the string with the letter "`f`".

f-strings in Python


```
1 val1 = "Data"
2 val2 = "Science"
3 print(f"{val1} {val2} is the study of {val1}.")
4
5
6 name = 'Tom'
7 age = 22
8 print(f"Hello... My name is {name} and I'm {age} years
    old.")
```

f-strings in Python

Quotation Marks in f-string: To use any type of quotation marks with the `f-string` in Python, we have to make sure that the quotation marks used inside the expression are not the same as quotation marks used with the `f-string`.

```
1 print(f"'M.Sc Data Science'")
2
3 print(f"""M.Sc "Data" Sciences""")
4
5 print(f'''M.Sc 'Data' Science''')
6
7 print(f'M.Sc "Data" Science')
```

f-strings in Python

Evaluate Expressions with f-Strings: We can also evaluate expressions with `f-strings` in Python. To do so, we have to write the expression inside the curly braces in `f-string` and the evaluated result will be printed.

```
CAT1 = 45
CAT2 = 40
Quiz1 = 8
Quiz2 = 7
DA = 9

print(f"Ram's  internal mark is: {CAT1*15/50 + CAT2
   *15/50 + Quiz1 + Quiz2 + DA} out of 60")
```

f-strings in Python

f-strings can be used in `input` to dynamically display the message and take the input from the user.

```
1 d
```

String Methods

- `replace()` – method replaces a string/character with another string/character

```
s = "Hello World!"
print(s.replace("H", "J")) #Prints "Jello World!"
print(s.replace("Hello", "Hai")) #Prints "Hai World
   !"
```

String Methods

- `split()` – this method splits the string into substrings if it finds the instance of the separator/character we have supplied and omitting that separator/character.

```
s = "Hello, World!"
print(s.split(",")) #Prints "['Hello', ' World!']"
print(s.split(" ")) #Prints "['Hello,', 'World!']"
print(s.split("o")) #Prints "['Hell', ' , W', 'rld
   !']"
```

String Methods

- To check if a certain character of phrase is present in a string or not, we can use the keywords `in` or `not in`.

```
txt = "There is a Rainbow in the Sky"
x = "ain" in txt
print(x) #Prints "True"
x1 = "rain" in txt
x2 = "Rain" in txt
x = "ain" not in txt
print(x) #Prints "False"
```

String Methods

- `strip()` – This method removes any whitespace from the beginning or the end of the string

```
s = "     Hello World!   "
print(s.strip()) #returns "Hello World!"
```

- `lower()` – This method returns the string in lower case
- `upper()` – This method returns the string in upper case
- `capitalize()` – This method converts the first character to upper case
- `title()` – This method returns the string in title case i..e, the first character of each word in upper case.
- `swapcase()` – This methods swaps cases, lower case by upper case and vice versa

String Methods


```
s = "hello World!"
s1 = s.lower()
print(s1) #prints "hello world!"
s2 = s.upper()
print(s2) #prints "HELLO WORLD!"
s3 = s2.capitalize()
print(s3) #prints "Hello world!"
s4 = s1.title()
print(s4) #prints "Hello World!"
s5 = s4.swapcase()
print(s5) #prints "hELLO wORLD!"
```

String Methods

- `islower()` – Returns true if the string is in lower case
- `isupper()` – Returns true if the string is in upper case
- `istitle()` – Returns true if the string is in title case
- `rstrip()` – Return the right trim version of the string
- `lstrip()` – Return the left trim version of the string

String Methods

- `count()` – Returns the number of times a specified value occurs in string
- `find()` – Searches for the specified for the string in the given string and returns the first position where it is found.
- `isalnum()` – Checks whether the string is alpha numeric
- `isalpha()` – Checks if all the characters in the string are alphabets
- `isdigit()` – Checks if all the characters in the string are digits

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable (mutable). Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable (immutable). Allows duplicate members.
- **Set** is a collection which is unordered and unindexed (mutable). No duplicate members.
- **Dictionary** is a collection which is ordered[1], changeable and indexed (mutable). No duplicate members.

[1]As of Python 3.7 and later dictionaries are ordered. In Python 3.6 and earlier dictionaries are unordered

Python Lists

In Pyhton a **list** is a collection which is ordered and changeable. In Python lists are written with square brackets.

```
myList = ["First", "Second", "Third", "Fourth", "Fifth"
    ]
print(myList)

print(myList[1]) #Prints the second element in the list
    . Index in Python starts from 0

print(myList[-1]) #Print the last element in the list.
    -1 refers to last element. -2 refers to second last
    element etc.
```

Python Lists


```
1 myFruits = ["apple", "banana", "cherry", "orange", "
    kiwi", "melon", "mango"]
2 print(myFruits[2:5]) # Prints the elements starts with
    index 2 and upto index 5 (not included)
3
4 print(myFruits[:4]) # Prints all the elements from
    starting first element to upto 4th element
5
6 print(myFruits[3:]) # Prints all the elements from
    starting with thrid index to last
```

Python Lists


```
print(myFruits[-4:-1]) # Prints the element from index -4 (included) to index -1 (excluded)

myFruits[1] = "grape" # Changes the entry in the index 1
print(myFruits)

"apple" in myFruits # Returns True if the apple is present in myFruits list otherwise returns False
```

Python Lists

Method	Description
`append()`	Adds an element at the end of the list
`clear()`	Removes all the elements from the list
`copy()`	Returns a copy of the list
`count()`	Returns the number of elements with the specified value
`extend()`	Add the elements of a list (or any iterable), to the end of the current list
`index()`	Returns the index of the first element with the specified value
`insert()`	Adds an element at the specified position
`pop()`	Removes the element at the specified position
`remove()`	Removes the item with the specified value
`reverse()`	Reverses the order of the list

Tuples

A **tuple** is a collection which is ordered and unchangeable or immutable. In Python tuples are written with round brackets.

```
1 myFruits = ("apple", "banana", "cherry")
2 print(myFruits)
```

Tuples

You cannot add or delete or change the elements of a tuple. To change them we need to convert a tuple into a list.

```
1 myList = list(myFruits)
2 print(myList)
3 myList[1] = "organge"
4 myFruits = tuple(myList)
5 print(myFruits)
```

Tuples

Create Tuple With One Item

To create a tuple with only one item, you have add a comma after the item, unless Python will not recognize the variable as a tuple.

```
1 myTuple = ("apple",)
2 print(type(myTuple))
3
4 #NOT a tuple
5 mytuple = ("apple")
6 print(type(mytuple))
```

Tuples

Remove Items from Tuple

One cannot remove items from a Tuple as Tuples are **unchangeable**. But we can delete completely the Tuple. using the `del` command.

```
1 myFruits = ("apple", "banana", "cherry")
2 del myFruits
3 print(myFruits) #this will raise an error because the
    tuple no longer exists
```

Tuples

Few more commands...

- Join two Tuples

```
tuple1 = ('First','Second','Third')
tuple2 = ('Fourth','Fifth','Sixith')
tuple3 = tuple1 + tuple2
print(tuple3)
```

- To know the number of times an element appears in a Tuple `tuple3.count('element')`
- To know the index of a particular entry in a Tuple `tuple3.index('Fourth')`

Tuples

Few more commands...

- Repeat the elements of a tuple

```
1  tuple1 = ('First','Second','Third')
2  tuple2 = tuple1 * 3
3  print(tuple2) # Prints the tuple1 3 times
```

Python Sets

- Sets are unordered collection of objects.
- In Python sets are written with curly brackets.
- We can also creates sets using the `set()` constructor by passing list of elements.
- Sets are unordered, so you cannot be sure in which order the items will appear.
- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.
- But we can loop through the elements of a set using `for` and membership function `in`

Python Sets


```
myfruits = {"apple", "banana", "cherry"}
print(myfruits)

for x in myfruits:
  print(x)

print("banana" in myfruits)

print("orange" in my fruits)
```

Python Sets

- To add an element(s) we can use `add('item')` or `update(['items'])` commands.
- To remove an element we can use either `remove('item')` or `discard('item')`
- `remove()` returns an error if the element does not exist in the Set.
- `discard()` does not return an error.
- We can also use `pop()` method. But as Sets are unordered, we will not know which element gets removed by using `pop()` command.

Python Sets


```
myfruits = {"apple", "banana", "cherry"}
myfruits.add("orange")
print(myfruits)

myfruits.update(["orange", "mango", "grapes"])
print(myfruits)

print(len(myfruits)) # Prints the no.of elements in the
    set

myfruits.remove("banana")
print(myfruits)
```

Python Sets


```
myfruits.discard("banana")
print(myfruits)

x = myfruits.pop()
print(x)
print(myfruits)

myfruits.clear()
print(myfruits)

myfruits = {"apple", "banana", "cherry"}
del myfruits
```

Python Sets

Method	Description
`add()`	Adds an element to the set
`clear()`	Removes all the elements from the set
`copy()`	Returns a copy of the set
`difference()`	Returns a set containing the difference between two or more sets
`difference_update()`	Removes the items in this set that are also included in another, specified set
`discard()`	Remove the specified item
`intersection()`	Returns a set, that is the intersection of two other sets
`intersection_update()`	Removes the items in this set that are not present in other, specified set(s)
`isdisjoint()`	Returns whether two sets have a intersection or not

Python Sets

Method	Description
`issubset()`	Returns whether another set contains this set or not
`issuperset()`	Returns whether this set contains another set or not
`pop()`	Removes an element from the set
`remove()`	Removes the specified element
`symmetric_difference()`	Returns a set with the symmetric differences of two sets
`symmetric_difference_update()`	inserts the symmetric differences from this set and another
`union()`	Return a set containing the union of sets
`update()`	Update the set with the union of this set and others

Python Sets


```
set1 = {"x","y","z"}
set2 = {1,2,3,4,5}

set3 = set1.union(set2) #union of set1 and set2
print(set3)

set4 = {"x","z",4,2,7,10}
set5 = set4.intersection(set1) #intersection of set4, set1
set6 = set4.intersection(set2) #intersection of set4, set2

print(set5)
print(set6)

set1.update(set2) # adds the elements of set2 to set1
```

Python Dictionaries

- A dictionary is a collection which is ordered[2], changeable and indexed.
- A dictionary is represented by a pair of curly braces {} in which enclosed are the “key: value” pairs separated by a comma.
- Python dictionary keys are immutable (which cannot be changed) data types that can be either strings or numbers.
- However, a key can not be a mutable data type, for example, a list.
- Keys are unique within a dictionary and can not be duplicated inside a dictionary.
- Key connects with the value, hence, creating a map-like structure.

[2]As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered

Python Dictionaries


```
myDict = {
    "brand": "HP",
    "btype": "Convertible",
    "byear": 2019
}
print(myDict)
```

Python Dictionaries

We can use the dict() constructor also to make a new dictionary:

```
myDict = dict(brand="HP", btype="Convertible", byear=2019)
# Please note that the keywords string literals without quotes
# We are using '=' in dict() constructor instead of colon (:) for assignment
print(myDict)
```

Python Dictionaries
Unique Keys

The keys in a dictionary have to be unique:

```
dictionary_unique = {"a": "Alpha", "b": "Beta", "g": "Gamma"}
print(dictionary_unique)

#Output: {'a': 'Alpha', 'b': 'Beta', 'g': 'Gamma'}

dictionary_unique = {"a": "Alpha", "b": "Beta", "g": "Gamma", "g": "Omega"}
print(dictionary_unique)

#Output: {'a': 'Alpha', 'b': 'Beta', 'g': 'Omega'}
```

Python Dictionaries
Accessing Keys and Values

If we want to access both the key, value pair, we could use `.items()` method, which will return a list of `dict_items` in the form of key, value tuple pairs.

```
ditems = dictionary_unique.items()
print(ditems)

print(myDict.items())
```

To access the keys of a dictionary:

```
print(myDict.keys())
```

To access the values of a dictionary:

```
print(myDict.values())
```

Python Dictionaries

we could even access a value by specifying a key as a parameter to the dictionary. To access the items of a dictionary we can either

- refer to its key name, inside square brackets
- we can use the `get()` method

```
x = myDict["type"]
print(x)
x = myDict.get("type")
print(x)
```

Python Dictionaries

To change the value of a specific item we refer to its key name: For example to change the "brand" to "Lenovo" we can use

```
myDict["brand"] = "Lenovo"
print(myDict)
```

Python Dictionaries
Removing Items

To remove an item from dictionary we can use `pop()` method and by passing the key name:

```
myDict = {
  "brand": "Lenovo",
  "btype": "Tablet",
  "byear": 2019
}
myDict.pop("btype")
print(myDict)
```

Python Dictionaries
Removing Items

The method `popitem()` removes the last inserted item (In Python versions before 3.7, this method removes a random item)

```
myDict = {
  "brand": "Lenovo",
  "btype": "Tablet",
  "byear": 2019
}
myDict.popitem()
print(myDict)
```

Python Dictionaries
Removing Items

The `del` keyword deletes the complete dictionary

```
del myDict
print(myDict)
```

`clear()` function empties the dictionary:

```
myDict = {
  "brand": "Lenovo",
  "btype": "Tablet",
  "byear": 2019
}
print(myDict)
myDict.clear()
print(myDict)
```

Python Dictionaries
Removing Items

The copy() command will be useful in making a copy of the dictionary already existing.

```
myDict = {
  "brand": "Lenovo",
  "btype": "Tablet",
  "byear": 2019
}
print(myDict)
myDict2 = myDict.copy()
print(myDict2)
```

Python Dictionaries
Nested Dictionaries

We can even create nested dictionaries

```
myDict = {
    "prod1": {
        "bname": "HP",
        "btype": "Computer",
        "byear": 2019
    },
    "prod2": {
        "bname": "Lenovo",
        "btype": "Tablet",
        "byear":2019
    },
    "prod3": {
        "bname": "Microsoft",
        "btype": "Convertible",
        "byear": 2020
    }
}
print(myDict)
```

Python Dictionaries
Nested Dictionaries

Another method

```
prod1 = {
        "bname": "HP",
        "btype": "Computer",
        "byear": 2019
    }
prod2 = {
        "bname": "Lenovo",
        "btype": "Tablet",
        "byear":2019
    }
prod3 = {
        "bname": "Microsoft",
        "btype": "Convertible",
        "byear": 2020
    }

myDict = {
    "prod1": prod1,
    "prod2": prod2,
    "prod3": prod3
}
print(myDict)
```

Python Dictionaries
Nested Dictionaries

Accessing the elements of a nested dictionary:

```
print(myDict["prod1"])
#Output: {'bname': 'HP', 'btype': 'Computer', 'byear': 2019}

myDict["prod1"]["bname"]
#Output: 'HP'

myDict["prod3"]["btype"]
#Output: 'Convertible'
```

Python Dictionaries
Looping through dictionaries


```
for x in myDict:
    print(x)    #Prints the key
    print(myDict[x])    #Prints the value
```

We can also use `values()` function to return the values of a dictionary:

```
for x in myDict.values():
    print(x)    #Prints the value
```

To loop through keys and values we can use `items()` function

```
for x,y in myDict.items():
    print(x, y)    #Prints the key, value
```

Python For Loop and Dictionary
Word Frequency

We can combine the `for` loop and `dictionary` data type in Python to count the word frequency in a string.

```
rand_str = 'Video provides a powerful way to help you prove your point. \
 When you click Online Video you can paste in the embed code for the video you want to add.'

word_freq = dict()
rand_str_word = str(rand_str).split()
for word in range(len(rand_str_word)):
    if rand_str_word[word] not in word_freq:
        word_freq[rand_str_word[word]] = 1
    else:
         word_freq[rand_str_word[word]] += 1
```

Python List Comprehension

Consider the following Mathematical Lists:

$$S = \{0, 1, 4, 9, 16, 25, 36, 49, 64, 81\};$$

$$V = \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\};$$

$$M = \{0, 4, 16, 36, 64\}$$

Loop Version


```
S = []
for x in range(10):
    S.append(x**2)
print(S)

V = []
for i in range(13):
    V.append(2**i)
print(V)

M = []
for x in S:
    if x%2 == 0:
        M.append(x)
print(M)
```

Python List Comprehension

- List comprehension is an important techniques using which we can create a list of numbers and dictionaries easily.
- List comprehension in Python is surrounded by brackets, but instead of the list of data inside it, you enter an expression followed by `for` loop and `if-else` clauses.
- A most basic form of List comprehension in Python is constructed as follows:
  `list_variable = [expression for item in collection]`
- The above expression generates elements in the list followed by a for loop over some collection of data which would evaluate the expression for every item in the collection.

Conditional List Comprehension

List Comprehension with an `if` condition:

```
listcomp = [expression for item in iterable if
   condition == True]
```

The above code is equivalent to the following `for` loop:

```
listcomp =[]
for item in iterable:
    if condition == True:
        listcomp.append(expression)
```

Conditional List Comprehension

List Comprehension with an `if...else` condition:

```
listcomp = [expression1 if condition == True else
   expression2 for item in iterable]
```

The above code is equivalent to the following `for` loop:

```
listcomp =[]
for item in iterable:
    if condition == True:
        listcomp.append(expression1)
    else:
        listcomp.append(expression2)
```

List Comprehension Version

The Python code with List Comprehension technique is:

```
S = [x**2 for x in range(10)]
# Output: S = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
V = [2**i for i in range(13)]
# Output: V = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512,
   1024, 2048, 4096]
M = [x for x in S if x % 2 == 0]
# Output M = [0, 4, 16, 36, 64]
M1 = [x if x%2 == 0 else x/2 for x in S]
# Output M1 = [0, 0.5, 4, 4.5, 16, 12.5, 36, 24.5, 64,
   40.5]
```

Advantages of List Comprehension

- Time-efficient and space-efficient than loops.
- Require fewer lines of code.
- Transforms iterative statement into a formula.

Nested List Comprehension

Consider the example of preparing a matrix

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Nested List Comprehension

Consider the example of preparing a matrix

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

the Python code using the `for` loop is:

```
M = []
for i in range(4):
    M.append([])
    for j in range(1,5):
        M[i].append(4*i+j)
print(M)
```

Nested List Comprehension

Consider the example of preparing a matrix

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

the Python code using the `for` loop and list comprehension is:

```
M = []
for i in range(4):
    M.append([])
    for j in range(1,5):
        M[i].append(4*i+j)
print(M)
```

```
M = [[4*i+j for j in range
   (1,5)] for i in range(4)]
print(M)
# Output: [[1, 2, 3, 4], [5,
   6, 7, 8], [9, 10, 11, 12],
   [13, 14, 15, 16]]
```

Python Set Comprehension


```
1 set_comprehension = {i**3 for i in range(10)}
2
3 print(set_comprehension)
4
5 for value in set_comprehension:
6     print(value,end=" ")
```

Python Dictionary Comprehension


```
dict_comprehension = {i: i**3 for i in range(10)}

for key,value in dict_comprehension.items():
    print(key,value)
```