Strings (and Multiple Inputs)

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Structured Data Types

- Looked at scalar data types - which have simple values
  - E.g. integer, real numbers, boolean
  - These data have no components/parts within them
- Python also has data types in which the data object is compound, i.e. a collection of scalar (structured) data items combined into one object
- In such data types, you can access the full data object, or its components / items
- In python there are some built-in structured types: Lists, Strings, Sets, Tuples, Dictionaries

- We will first discuss lists and strings (strings often not considered in the same category)
- Will then discuss how to take multiple inputs from user

Strings

Strings

- Strings are within double quotes ("") or single quotes ('')
- A sequence of ascii chars - alphabets, numbers, special characters, etc all allowed (except " or ' - as they are delimiters)
- New line is a character represented by '\n'
- Strings can be assigned to variables, like numbers

```
s = "Hello, world"
name = "Pankaj Jalote"
```

- Multiple line strings are assigned with three quotes, like

```
addr = '''IIIT Delhi
        Okhla, phase 3'''
```

- In multiline - newline is treated as a character

Strings are compound objects

- Strings are a sequence of characters, so can access each char
- String also support index and slicing like lists.
- In s = "Hello World", s[0] is 'H', s[1] is 'e' ….
- Like in list, can loop over a string with characters as items

```
for c in str:
    print(c)
```

Slicing Strings

- Like lists, can slice strings, E.g.

  s = "Hello, World"

  s[1:6] # will give 'ello,'

- Slicing from start: s[:4] # will be 'Hell'
- Slicing till the end: s[6:] # will be ' World'
- Negative indexing, i.e. from the end

  s [-1] # gives last char, i.e. 'd'

  s [-5:] # 'World'; s[-5:-2] # 'Wor'

Operations on Strings

Functions on strings - some similar to list

- len(s) # returns the length of string, i.e. the no of characters

  str1 = “Hello” # len(str1) returns 5

- in: (Membership Testing) Returns true if a character or substring exists in the given string.

  "He" in str1 # true

- not in: Returns true if a character or substring does not exist in the given string.

  “h” not in str1 # True

Operations +/*: Concatenating / replicating

- Joining, i.e. concatenating strings is a common need, + operation, e.g.

```
first, last = "Pankaj", "Jalote"
name1 = first+last  # this will not have a blank between
name2 = first + " " + last   # now have a blank
name3 = last + ", " + first  # last name first, with comma
```

- * operator for replication

```
s = "Hello"
res = s*4 # Now res = “HelloHelloHelloHello”
```

Quiz-Numerical

Consider the string given below and answer the question that follows.

```
my_string="""CSE-101
IP
"""
```

What value will the function len(my_string) return?

Quiz-Numerical(Solution)

Consider the string given below and answer the question that follows.

```
my_string="""CSE-101
IP
"""
```

What value will the function len(my_string) return?

Solution: 11

Explanation: my_string is a multiple line string and each new line will also be considered as a separate character

String methods: Splitting strings

- A string often has sub-strings separated by blank, special char, etc. e.g. a sentence with words
- Often you need to get the words in a string
- The split() method splits a string and creates a list of substrings. The argument supplied is used as the separator. Splits on whitespaces(spaces,tabs, newline etc..) if no argument is supplied
- Example : s = “CSE101, Intro to Programming”

```
s.split()  # separator is any whitespace; returns a list

returns ['CSE101,', 'Intro', 'to', 'Programming']

s.split(‘,’) # splits using “,”;

returns ['CSE101', ' Intro to Programming']
```

String joining

- Often need to join multiple strings (e.g. in a list), with some "separator" between them to form one large string
- Items must all be of string type

  sep.join(<list of strings>)

  ",".join(["one", "two"]) will return "one,two"
- Can do the above with + operation also
- E.g. Convert a list of integers into a string with blanks in between

  l = [23, 34, 45, 56]

  ls = [str(i) for i in l]

  " ".join(ls)
- Similar to + operation, but operations like above easier with join

Quiz – Text

What would be the output of the code given below?

```
lst = 'Introduction to programming 1 2 3 Hello World'.split()
print(lst[6][-4:-1])
```

Quiz – Text

What would be the output of the code given below?

```
lst = 'Introduction to programming 1 2 3 Hello World'.split()
print(lst[6][-4:-1])
```

Output: ell

lst[6] refers to the 7th element of lst i.e. ‘Hello’

A slice starting with 4th last character to 2nd last character

Strings are immutable

- Strings are immutable - cannot change a string object
-
- So, cannot change an element, i.e. cannot do:

  S[0] = "H" # this is not allowed - give error

- Ops like remove(), del(), pop() are not permitted for str - error
- (While immutable, compiler optimizes and "interns" some strings - i.e. reuses those objects - as done in integers till 256).

- While cannot change a string, can create new strings specifying changes existing strings through some operations

New string – modified existing strings

- Cannot change a string, but some methods available to create a new string with changes w.r.t. an existing string object, s

  s.lower() # gives a str which is s with all lower case

  s.upper() # gives a str which is s with all capitals, upper case

  s.strip() # str without any blanks at the beginning or end

  s.replace("str1", "str2") # string in which str1 in s is changed to str2

- Note there is no change of these operations on original string s
- All these operations therefore return a string (new) - so remove() in lists changes the existing list, while replace() in str creates a new str

Other Methods

- Huge number of methods available - all return new str - do not change original string (https://www.geeksforgeeks.org/python-string-methods/)
- These make string processing very easy in python
- Some we have seen, some others are:
  - s.count("str") # returns how many times "str" occurs in s
  - s.find("str") # index from where str is found in s; -1 if not found
  - s.isalpha() # true if all chars are alphabets
  - s.isalnum() # returns True if all are alpha numeric
  - s.isdigit() # True if all are digits, else false
  - s.capitalize() # capitalizes the first char and converts all other characters in the string to lowercase
  - s.title() # Capitalizes the first letter of every word in the string and converts the rest in lowercase
  - s.isupper() and s.islower(). # Returns true if s is an uppercase string and lowercase string respectively.
  - s.startswith(str) and s.endswith(str). # Used for checking suffixes and prefixes

Quiz-MCQ

Given a string, remove all non-alphabetic characters from it. For example if my_string=”intro## 5toprogramming”, my_string should be converted to “introtoprogramming”.

Which of the following code snippets will perform the task mentioned above?

A)

```
my_string="intro## ,.5toprogramming"

for s in my_string:
   if not s.isalpha():
      my_string=my_string.replace(s, "")
```

B)

```
my_string="intro## ,.5toprogramming"

for s in my_string:
   if s.isalpha():
      my_string=my_string.replace(s, "")
```

C)

```
my_string="intro## ,.5toprogramming"

for s in my_string:
   if not s.isalpha():
      my_string.replace(s, "")
```

D)

```
my_string="intro## ,.5toprogramming"

for s in my_string:
   if  s.isalpha():
      my_string.replace(s, "")
```

Quiz-MCQ(Solution)

Given a string, remove all non-alphabetic characters from it. For example if my_string=”intro## 5toprogramming”, it should be converted to “introtoprogramming”.

Which of the following code snippets will perform the task mentioned above?

A)

```
my_string="intro## ,.5toprogramming"

for s in my_string:
    if not s.isalpha():
        my_string=my_string.replace(s, "")
```

B)

```
my_string="intro## ,.5toprogramming"

for s in my_string:
    if s.isalpha():
        my_string=my_string.replace(s, "")
```

C)

```
my_string="intro## ,.5toprogramming"

for s in my_string:
    if not s.isalpha():
        my_string.replace(s, "")
```

D)

```
my_string="intro## ,.5toprogramming"

for s in my_string:
    if  s.isalpha():
        my_string.replace(s, "")
```

Option A is correct

Other Methods

- Reverse words in a given string
- Recall:
  - s.reverse() : Reverses the order of items in list

Another way to reverse a list?

Use Slicing

s[::-1]

```
# reverse words in a stmt
s = "Introduction to programming"

words = s.split(' ') #creates a list
words.reverse() # reverse the list
reverse = ' '.join(words) #making one str
print(reverse) # “programming to Introduction”
```

Escape Characters

- To give " or ' in string, use escape char \, i.e. have "Hello \" hi \' xx"
- For a newline we have \n
- Other escape characters:
  - \' : single quote
  - \t : tab
  - …

String Comparison

- Comparison takes place character by character
- If corresponding chars at a position satisfy condition, move to the next position and compare. Otherwise, return False.
- Comparisons are case-sensitive
- Unicode values of characters are compared (i.e. ord(c))
- Let str1 = “Hello”, str2 = “How”, str3 = “hello”, str4 = “Hello”, str5=”Hello World”

```
print(str1==str3) # False
print(str1==str4) # True
print(str1!=str3) # True
print(str1!=str4) # False
print(str1<str2) # True
print(str1>str2) # False
print(str1<=str3) # True
print(str1>=str3) # False
print(str1<str5) # True
print(str1>str5) # False
```

Quiz – Text

What would be the output of the following code:

```
str1 = "Introduction"
chunks = str1.split('o') # splits a str using the separator;
for i in chunks:
    print(i, end=' ')  # with end = ' ', a new line is avoided
```

Quiz – Alphanumeric

What would be the output of the following code:

```
str1 = "Introduction"
chunks = str1.split('o')
for i in chunks:
    print(i, end=' ')
```

Output: Intr ducti n

Explanation: The string is split using ‘o’ as the separator and all the chunks obtained after the split and printed with space in between.

Taking Multiple Inputs

- Suppose you want to give as input values of a, b, c (int) together
- In input(), python takes whatever input is given as string
- For int, float, bool - we can convert this string (If the input is not an integer/real number, conversion fails)
- If we give many values, .. entire input is treated as one string

```
X = input("Give: ") # input given: 11 22 33 44

print(x) # will give "11 22 33 44"
```

- Converting it to int() will give an error
- With string and list operations we can get this as a list of values

Multiple integer (or float) inputs

- Split the input string - this gives you a list of substrings which were separated by "blank" (you can also specify the separator)

```
x = input("Give: ").split() # input given: 11 22 33
print(x) # will print ["11",  "22",  "33"]
```

- To get list of integers from list of strings, list comprehension can be used

```
int_lst = [int(num) for num in x] # lst is now a list of integers
```

- We can combine them to get input a list:

```
int_lst = [int(num) for num in input("Give: ").split()]
```

- We can now use this list to get the values of a, b, c

```
a, b, c = int_lst
```

- Can combine them in one statement:

```
a, b, c = [int(num) for num in input("Give three numbers:").split()]
```

Quiz : Single Correct

What is the output of the code given?

```
lst = 'Introduction 12345 to 678 programming'
res = [i for i in (int(i) for i in lst if i.isdigit())]
print(res)
```

A. Error
B. [1, 2, 3, 4, 5]
C. [1,2,3,4,5,6,7,8]
D. [6, 7, 8]

Quiz : Single Correct

What is the output of the code given?

```
lst = 'Introduction 12345 to 678 programming'
res = [i for i in (int(i) for i in lst if i.isdigit())]
print(res)
```

A. Error
B. [1, 2, 3, 4, 5]
C. [1, 2, 3, 4, 5, 6, 7, 8]
D. [6, 7, 8]

Explanation: We first check for digits, finally printing them out as a list.

Summary – Strings

- Strings are immutable, i.e. can access a string item, but cannot change the item
- Can slice a string to get substrings - from start or end; can loop over string
- Functions: len(), in, not in, + , *
- Can split strings into a list of items using s.split()
- Can join a list of strings to form one using join()
- String operations (return a new string): lower(), upper(), replace(), count(), find(), isdigit(), ...
-
- Using string operations and list comprehension and conversion, we can now extract multiple inputs from an input string read by input function

Extras

Practice Exercises

- From a list of numbers, form list of those numbers that have a digit (say 5) in them (can use list comprehension)
- Given a list of strings, create a list which contains strings from this list which end with "a"

Exercise – Lists

- For a list determine the frequency distribution of different items
- There are many ways: Maintain two lists: unique_items, frequency, and then populate them
  - Take each item, traverse the rest of the list, count and mark items
  - Sort the list - then count successive items till a different item
  - Traverse the list - count the frequency of the item, remove all the items
  - …
- You can use time function to see which is most efficient
- Generate random lists of 1000 items

Exercise

- Split using a few different separators given in a sep_str (eg: sep_str = " , . ;?").
- Write a program to do this (cannot use re)
- Share code with TAs

Traversing Strings

Using For loop

```
s="Programming"
for i in s:
    print(i, end="-")
for i in range(len(s)):
    print(s[i], end="_")
```

Output :

```
P-r-o-g-r-a-m-m-i-n-g-
P_r_o_g_r_a_m_m_i_n_g_
```

Using While loop

```
s = "Programming"
i = 0
while i < len(s):
    print(s[i]+"_", end="")
    i = i + 1
```

Output :

```
P_r_o_g_r_a_m_m_i_n_g_
```