Unit 5 – Strings – Dictionaries

Course Learning Objectives

- ❖ Learn how Python inputs strings
- ❖ Understand how Python stores and uses strings
- ❖ Perform slicing operations on strings
- ❖ Traverse strings with a loop
- ❖ Compare strings and substrings
- ❖ Understand the concept of immutable strings
- ❖ Understanding string functions.
- ❖ Understanding string constants

Module – 1

5.1 Definition

In python, consecutive sequence of characters is known as a string. Strings are immutable and amongst the most popular types in Python. In python, String literals are surrounded by single or double or triple quotation marks.

‘RGUKT’ is same as “RGUKT”

5.1.1 Declaring & Initializing String Variables

Assigning a string to a variable is done with the variable name followed by an equal sign and the string

```
>>> s1 = “This is a string one”

>>> s2 = “This is a string two”

>>> s3=” “                        #empty string

>>> type(s1) <class 'str'>
```

- Multi-line strings can be denoted using triple single or double quotes, ''' or """.
- Even triple quotes can be used in Python but generally used to represent **multiline strings** or **docstrings**.

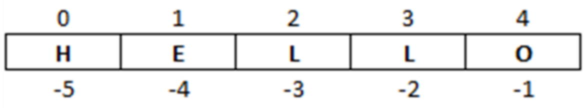
```
>>> s4 = ‘’’a multiline

            String’’’
```

```
>>> v = ''' Python is a programming language.

Python can be used on a server to create web applications. '''
>>> print(v)
 Python is a programming language.

Python can be used on a server to create web applications.
```

5.1.2 How to access characters in String

- We can access individual characters using indexing and a range of characters using slicing operator **[ ]** or **The Subscript Operator**
- Index starts from 0. Trying to access a character out of index range will raise an Index Error.
- The index must be an integer. We can't use float or other types; this will result into Type Error.
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and soon.
- We can access a range of items in a string by using the slicing operator with colon **[:]**.

S=”hello”



Important points about accessing elements in the strings using index

- Positive index helps in accessing the string from the beginning
- Negative index helps in accessing the string from the end.
- Index 0 or –ve n (where n is length of the string) displays the first element.
  Example: A[0] or A[-5] will display “H”

- Index -1 or (n-1) displays the last element.
  Example: A[-1] or A[4] will display “O”

Note: Python does not support character data type. A string of size 1 can be treated as characters.
If we try to access index out of the range or use decimal number, we will get errors.

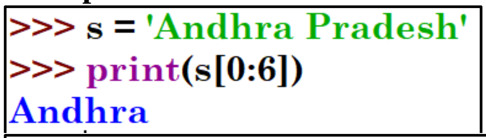
Accessing sub strings/Slicing Operator:
To access substrings, use the square brackets for slicing, using slice operator, along with the index or indices to obtain your substring.

Syntax:

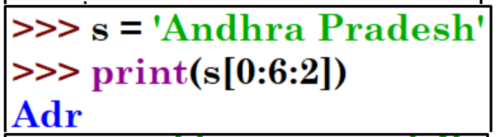
stringname[start:end:step]

start – starting point of the index value (default value: 0)
end or stop – ending point of the index value (default value: len(stringname))
step- increment of the index values (default value: 1)

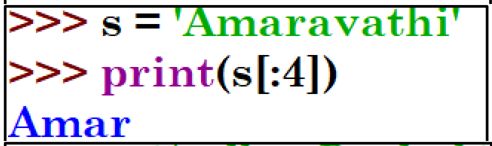
Examples:



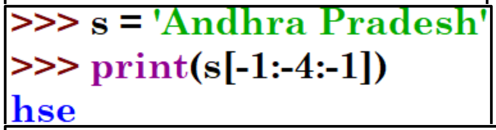
```
>>> s = 'Andhra Pradesh'
>>> print(s[0:6])
Andhra
```



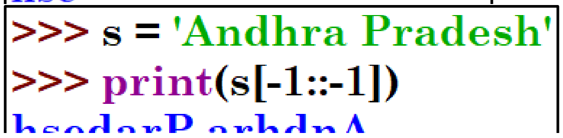
```
>>> s = 'Andhra Pradesh'
>>> print(s[0:6:2])
Adr
```



```
>>> s = 'Amaravathi'
>>> print(s[:4])
Amar
```



```
>>> s = 'Andhra Pradesh'
>>> print(s[-1:-4:-1])
hse
```



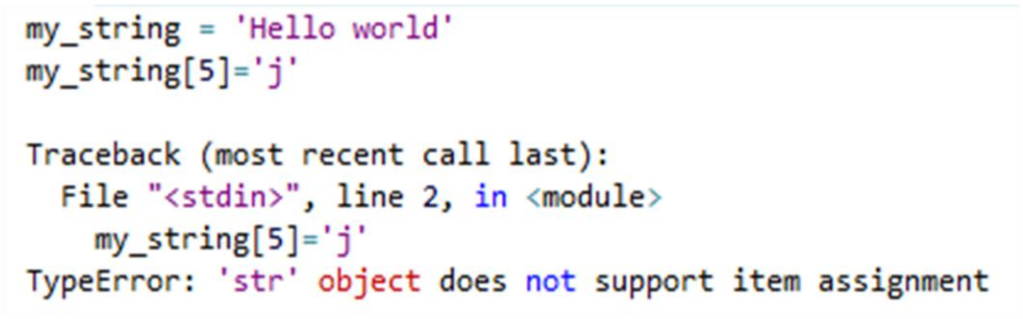
```
>>> s = 'Andhra Pradesh'
>>> print(s[-1::-1])
hsedarP arhdnA
```

5.1.3 How to change or delete a string?

Strings are immutable

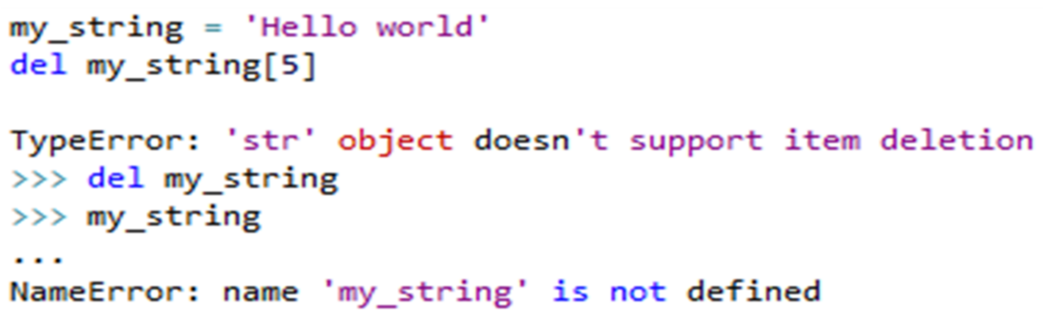
Strings are immutable means that the contents of the string cannot be changed after it is created.

Let us understand the concept of immutability with help of an example. We can simply reassign different strings to the same name.



```
my_string = 'Hello world'
my_string[5]='j'

Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
    my_string[5]='j'
TypeError: 'str' object does not support item assignment
```

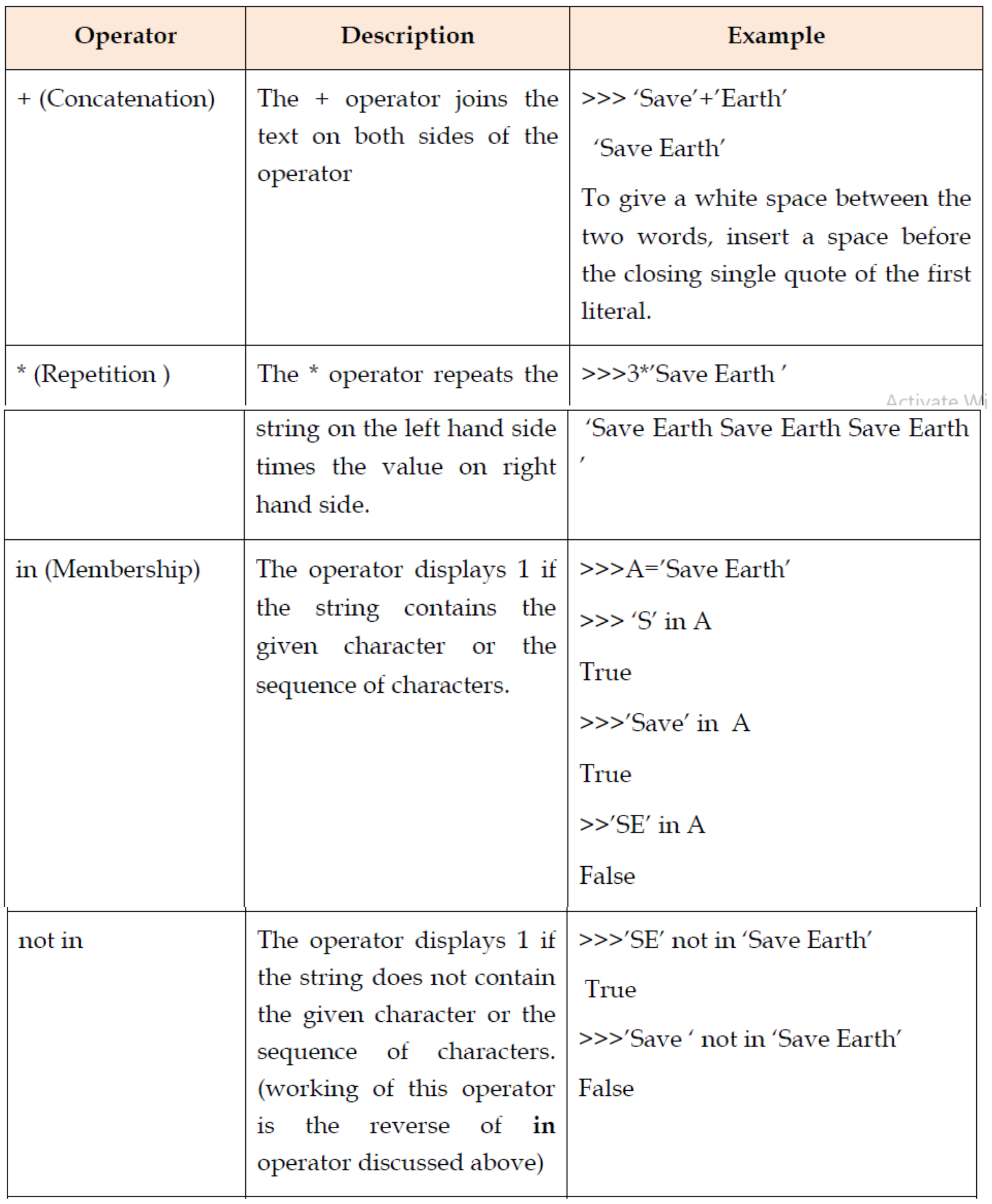
We cannot delete or remove characters from string, deleting the string is possible use the keyword **del**.



```
my_string = 'Hello world'
del my_string[5]

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

5.2. Python String Operations

There are many operations that can be performed with string which makes it one of the most used data types in python



Operator	Description	Example
+ (Concatenation)	The + operator joins the text on both sides of the operator	>>> 'Save'+'Earth' 'Save Earth' To give a white space between the two words, insert a space before the closing single quote of the first literal.
* (Repetition)	The * operator repeats the string on the left hand side times the value on right hand side.	>>>3*'Save Earth ' 'Save Earth Save Earth Save Earth '
in (Membership)	The operator displays 1 if the string contains the given character or the sequence of characters.	>>>A='Save Earth' >>> 'S' in A True >>>'Save' in A True >>'SE' in A False
not in	The operator displays 1 if the string does not contain the given character or the sequence of characters. (working of this operator is the reverse of **in** operator discussed above)	>>>'SE' not in 'Save Earth' True >>>'Save ' not in 'Save Earth' False

Comparing Strings: Python allows you to compare strings using relational (or comparison) operator such as

- = =,! =, >, <, < =, >=,etc.
- == if two strings are equal, it returns True
- != or <> if two strings are not equal, it returns True
    - ➢ if first string is greater than the second, it returns True
- < if second string is greater than the first ,it returns True
- >= if first string is greater than or equal to the second, it returns True
- <= if second string is greater than or equal to the first, it returns True

Note:

- *These operators compare the strings by using the lexicographical order i.e using ASCII values of the characters.*
- *The ASCII values of A-Z is 65 -90 and ASCII code for a-z is 97-122 .*

Logical Operators on String in Python: Python considers empty strings as having boolean value of 'false' and non-empty string as having boolean value of 'true'.

Let us consider the two strings namely str1 and str2 and try boolean operators on them:

```
str1 = 'IIIT'
str2 = ''

print(str1) #returns IIIT
print(repr(str1)) #returns 'IIIT'

#logical operators on strings

print(str1 or str2) #returns IIIT
print(str1 and str2) #returns empty string
print(repr(str1 and str2)) #returns empty string in quotes

print(not str1) #returns False
print(not str2) #returns True
```

Output:

```
IIIT
'IIIT'
IIIT

''
False
True
```

5.3 Escape sequence characters

- To insert characters that are illegal in a string, use an escape character.
- Escape sequence characters or non-printable characters that can be represented with backslash notation.
- An escape character gets interpreted; in a single quoted as well as double quoted strings

An example of an illegal character is a double quote inside a string that is surrounded by double quotes

Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Vikings" from the north."
```

#You will get an error if you use double quotes inside a string that are surrounded by double quotes:

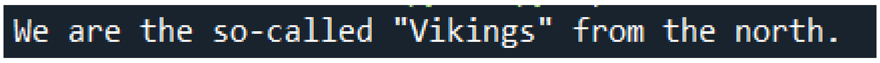
```
    txt = "We are the so-called "Vikings" from the north."
                                ^
SyntaxError: invalid syntax
```

To fix this problem, use the escape character \":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the north."
print(txt)
```



```
We are the so-called "Vikings" from the north.
```

Other escape characters used in Python

Escape Character	Description	Example	Result

\'	Single Quote	txt = 'It\'s alright.'
		print(txt)
			It's alright.

\\	Backslash	txt = "This will insert one \\ (backslash)." print(txt)	This will insert one \ (backslash).

\n	New Line	txt = "Hello\nWorld!"
		print(txt)
			Hello
			World!

\r	Carriage Return	txt = "Hello\rWorld!"
		print(txt)
			Hello
			World!

\t	Tab	txt = "Hello\tWorld!"
		print(txt)
			Hello World!

\b	Backspace	#This example erases one character (backspace):
		txt = "Hello \bWorld!"
		print(txt)
			HelloWorld!

\ooo	Octal value	#A backslash followed by three integers will result in a octal value:
		txt = "\110\145\154\154\157"
		print(txt)
			Hello

\xhh	Hex value	#A backslash followed by an 'x' and a hex number represents a hex value:
		txt = "\x48\x65\x6c\x6c\x6f"
		print(txt)
			Hello

Module – 2

5.4 Iterating Through String/Traversing a string

Traversing a string means accessing all the elements of the string one after the other by using the subscript. A string can be traversed using: for loop or while loop.

Example:

```
"""
Python Program:
 Using for loop to iterate over a string in Python
"""
string_to_iterate = "RGUKT"
for char in string_to_iterate:
   print(char)
```

Example:

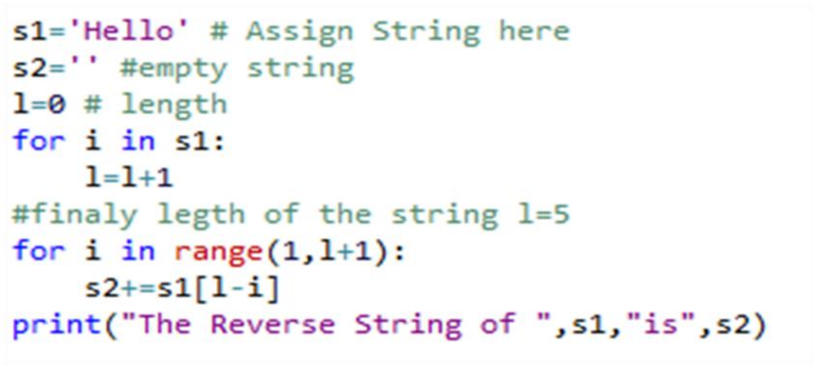
```
"""
Python Program:
 Using for loop to iterate using index of a string in Python
"""
string_to_iterate = "RGUKT"
for i in range(len(string_to_iterate)):
   print(string_to_iterate[i])
```

Example:

```
"""
Python Program:
 Using while loop to iterate using index of a string in Python
"""
string_to_iterate = "RGUKT"
i=0
while i< len(string_to_iterate):
   print(string_to_iterate[i])
   i=i+1
```

Example:

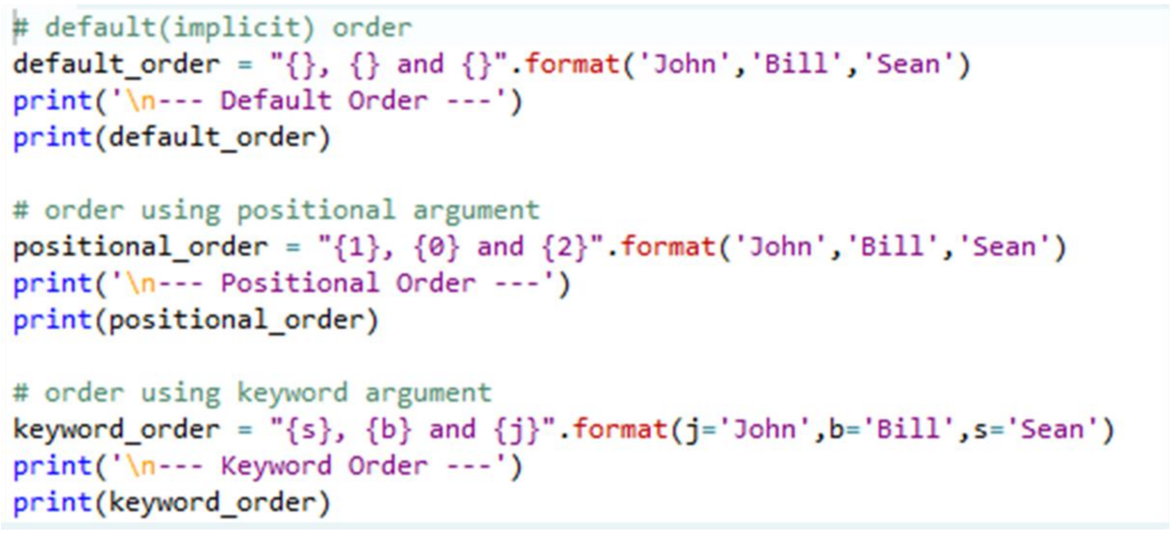
Python program to print reverse string using for loop



```
s1='Hello' # Assign String here
s2='' #empty string
l=0 # length
for i in s1:
    l=l+1
#finaly legth of the string l=5
for i in range(1,l+1):
    s2+=s1[l-i]
print("The Reverse String of ",s1,"is",s2)
```

5.5 The format() Method for Formatting Strings

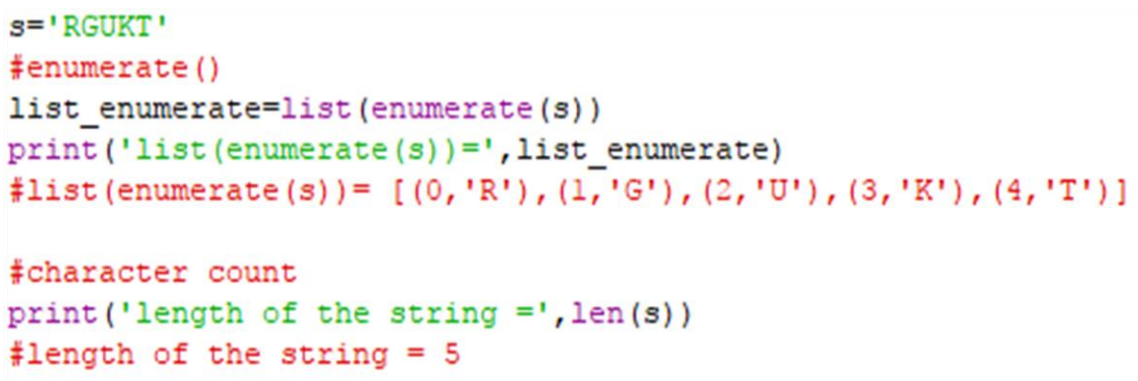
- The format() method that is available with the string object is very versatile and powerful in formatting strings.
- Format strings contains curly braces {} as placeholders or replacement fields which gets replaced.
- We can use positional arguments or keyword arguments to specify the order.



```
# 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)
```

5.6 Built-in functions to Work with Python Strings

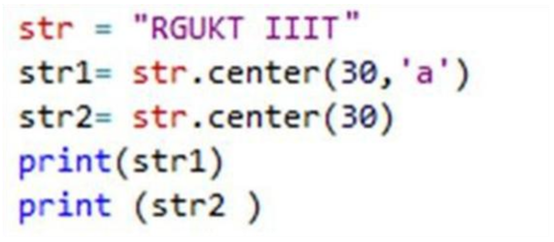
- Various built-in functions that work with sequence, works with string as well.
- The **enumerate()** function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.
- The **len()** function returns the length (number of characters) of the string.



```
s='RGUKT'
#enumerate()
list_enumerate=list(enumerate(s))
print('list(enumerate(s))=',list_enumerate)
#list(enumerate(s))= [(0,'R'),(1,'G'),(2,'U'),(3,'K'),(4,'T')]

#character count
print('length of the string =',len(s))
#length of the string = 5
```

- The method **center()** makes str centred by taking width parameter into account. Padding is specified by parameter ***fillchar***. Default filler is a space.

  Syntax: str.center(width[, fillchar])



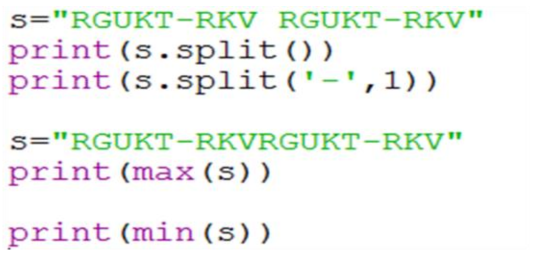
```
str = "RGUKT IIIT"
str1= str.center(30,'a')
str2= str.center(30)
print(str1)
print (str2 )
```

- The function **max()** returns the max character from string str according to ASCII value.in first print statement y is max character, because ASCII code of "y" is 121. In second print statement "s" is max character, ASCII code of "s" is 115.

  Syntax :- max(str)

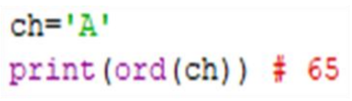
- The function **min()** returns the min character from string str according to ASCII value.

  Syntax :- min(str)



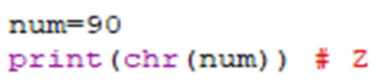
```
s="RGUKT-RKV RGUKT-RKV"
print(s.split())
print(s.split('-',1))

s="RGUKT-RKVRGUKT-RKV"
print(max(s))

print(min(s))
```

- The **ord()** function returns ASCII code of the character.



```
ch='A'
print(ord(ch)) # 65
```

- The **chr()** function returns character represented by a ASCII number.

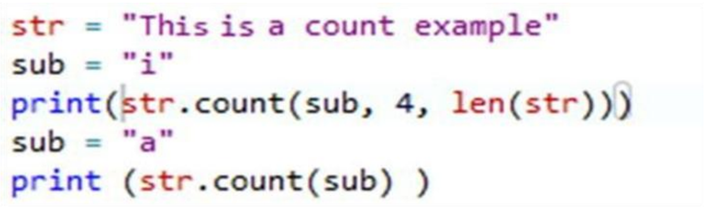


```
num=90
print(chr(num)) # Z
```

5.7 String Methods

- **count():** The method **count()** returns the number of occurrence of Python string *substr* in string *str*. By using parameter *start* and *end* you can give slice of *str*. This function takes 3 arguments, substring, beginning position (by default 0) and end position (by default string length). Return Int Value

  Syntax: str.count(substr [, start [, end]])

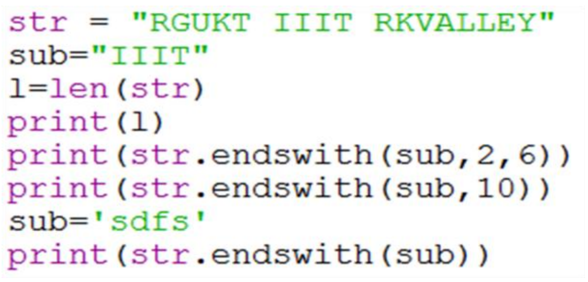


```
str = "This is a count example"
sub = "i"
print(str.count(sub, 4, len(str)))
sub = "a"
print (str.count(sub) )
```

- **endswith(“string”, beg, end):** This function returns true if the string ends with mentioned string(suffix) else return false. Return Bool Value

  The use of *start* and *end* to generate slice of Python string str.

  Syntax: str.endswith(suffix[, start[, end]])



```
str = "RGUKT IIIT RKVALLEY"
sub="IIIT"
l=len(str)
print(l)
print(str.endswith(sub,2,6))
print(str.endswith(sub,10))
sub='sdfs'
print(str.endswith(sub))
```

- **startswith(“string”, beg, end):** This function returns true if the string begins with mentioned string(prefix) else return false.

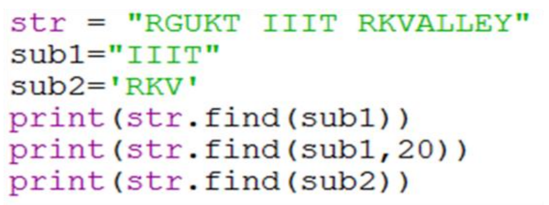
```
s = 'Amaravathi, Tirupathi'
i = s.startswith("Amar") #checks from the beginning of the string and returns result
print(i) #displays True

i = s.startswith("Amar", 10, 20) #checks from 10th index and returns result
print(i) #displays False
```

- **find(“string”, beg, end):** This function is used to find the position of the substring within a string. It takes 3 arguments, substring , starting index(by default 0) and ending index(by default string length).
  - It returns “-1” if string is not found in given range.
  - It returns first occurrence of string if found.

  Return the lowest index in S where substring sub is found

  If given Python string is found, then the **find()** method returns its index. If Python string is not found then -1 would be returned.

  Syntax : str.find(str, beg=0 end=len(string))



```
str = "RGUKT IIIT RKVALLEY"
sub1="IIIT"
sub2='RKV'
print(str.find(sub1))
print(str.find(sub1,20))
print(str.find(sub2))
```

- **rfind(“string”, beg, end):** This function is similar to find(), but it returns the position of the last occurrence of sub string.

```
s = "Amaravathi, Tirupathi"
i = s.rfind("thi") #checks entire string and returns the last occurrence of substring i.e., 18
print(i)
```

- **replace():** This function is used to replace the substring with a new substring in the string. This function has 3 arguments. The string to replace, new string which would replace and max value denoting the limit to replace action (by default unlimited).

```
#string.replace(oldstring, newstring)
st="RGUKT RKV"
n=st.replace('RKV','ONG')
print(n)
```

- The method **isalnum()** is used to determine whether the Python string consists of alphanumeric characters, false otherwise

  Syntax :- **str.isalnum()**

- The method **isalpha()** return true if the Python string contains only alphabetic character(s), false otherwise.

  Syntax :- **str.isalpha()**

- The method **isdigit()** return true if the Python string contains only digit(s),false otherwise.

  Syntax :- **str.isdigit()**

- The method **islower()** return true if the Python string contains only lower cased character(s), false otherwise

  Syntax :- **str.isdigit()**

```
s='rgukt1234'
print(s.isalnum())
s1='#$@#%'
print(s1.isalnum())
s2='PinCode'
print(s2.isalpha())
s3='rgukt1234'
print(s3.isalnum())
s4='1234'
print(s3.isdigit())
s5='rgukt1234'
print(s5.isdigit())
```

```
s='rguktrkv'
print(s.islower())
s1='pin343223'
print(s1.islower())
s2='PinCode'
print(s2.islower())
```

- The method **isspace()** return true if the Python string contains only white space(s).

  Syntax :- **str.isspace()**

- The method **istitle()** return true if the string is a titlecased

  Syntax :- **str.istitle()**

- The method **isupper()** return true if the string ontains only upper cased character(s), false otherwise.

  Sytax:- **str.isupper()**

- The method **ljust(),** it returns the string left justified. Total length of string is defined in first parameter of method *width*. Padding is done as defined in second parameter *fillchar* .(default is space)

  Syntax : - **str.ljust(width[, fillchar])**

```
s="  "
print(s.isspace())

s1="pin 2342"
print(s1.isspace())

s2="rgukt rkv"
print(s2.istitle())

s3="Rgukt rkv"
print(s3.istitle())

s4="Rgukt Rkv"
print(s4.istitle())
```

```
s5="RGUKT"
print(s5.isupper())

s6="RGUKT123"
print(s6.isupper())

s7="rGUKT"
print(s7.isupper())

s="rgukt"
print(s.ljust(10,"k"))
print(s.ljust(10))
print(s.ljust(3,"k"))
```

- In above example you can see that if you don't define the *fillchar* then the method **ljust()** automatically take space as *fillchar*.

- The method **rjust(),** it returns the string right justified. Total length of string is defined in first parameter of method *width*. Padding is done as defined in second parameter *fillchar* .(default is space)

Syntax:- **str.rjust(width[, fillchar])**

- This function **capitalize()** first letter of string.

  Syntax:- **str.capitalize()**

- The method **lower()** returns a copy of the string in which all case-based characters have been converted to lower case.

  Syntax:- **str.lower()**

- The method **upper()** returns a copy of the string in which all case-based characters have been converted to upper case.

  Syntax:- **str.upper()**

- The method **title()** returns a copy of the string in which first character of all words of string are capitalised.

  Syntax:- **str.title()**

- The method **swapcase()** returns a copy of the string in which all cased based character swap their case

  Syntax:- **str.swapcase()**

```
s = "APJ Kalam is GREAT Scientist"

l = s.lower() #returns a new string with all lower case letters in it
print(l)

u = s.upper() #returns a new string with all upper case letters in it
print(u)

sc = s.swapcase() #returns a new string 'apj kALAM IS great sCIENTIST'
print(sc)

t = s.title() #returns a new string 'Apj Kalam Is Great Scientist'
print(t)

c = s.capitalize() #returns a new string 'Apj kalam is great scientist'
print(c)
```

- This method **join()** returns a string which is the concatenation of given sequence and string as shown in example.

  seq = it contains the seqeunce of separated strings.

  str = it is the string which is used to replace the separator of seqcence

  Syntax:- **str.join(seq)**

- The method **lstrip()** returns a copy of the string in which specified char(s) have been stripped from left side of string. If char is not specified then space is taken as default.

  Syntax : - **str.lstrip([chars])**

- The method **rstrip()** returns a copy of the string in which specified char(s) have been stripped from right side of string. If char is not specified then space is taken as default.

  Syntax : - **str.rstrip([chars])**

- The method **strip()** returns a copy of the string in which specified char(s) have been stripped from both side of string. If char is not specified then space is taken as default.

  Syntax : - **str.strip([chars])**

- The method **split()** returns a list of all words in the string, delimiter separates the words. If delimiter is not specified then whitespace is taken as delimiter, paramter num

Syntax :- **str.split("delimiter", num)**

Module 3 -- Dictionaries

A **dictionary** is like a list, but more general. In a list, the index positions have to be integers; in a dictionary, the indices can be (almost) any type.

You can think of a dictionary as a mapping between a set of indices (which are called **keys**) and a set of values. Each key maps to a value. The association of a key and a value is called a **key-value pair** or sometimes an **item**.

As an example, we'll build a dictionary that map from English to Spanish words, so the keys and the values are all strings.

The function **dict** creates a new dictionary with no items. Because **dict** is the name of a built-in function, you should avoid using it as a variable name.

```
>>> eng2sp = dict()
>>> print (eng2sp)
{}
```

The curly brackets, {}, represent an empty dictionary. To add items to the dictionary, you can use square brackets:

```
>>> eng2sp['one'] = 'uno'
```

This line creates an item that maps from the key 'one' to the value 'uno'. If we print the dictionary again, we see a key-value pair with a colon between the key and value:

```
>>> print (eng2sp)
{'one': 'uno'}
```

This output format is also an input format. For example, you can create a new dictionary with three items:

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

But if you print eng2sp, you might be surprised:

```
>>> print (eng2sp)
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

The order of the key-value pairs is not the same. In fact, if you type the same example on your computer, you might get a different result. In general, the order of items in a dictionary is unpredictable.

But that's not a problem because the elements of a dictionary are never indexed with integer indices. Instead, you use the keys to look up the corresponding values:

```
>>> print (eng2sp['two'])
'dos'
```

The key **'two'** always maps to the value 'dos' so the order of the items doesn't matter.
If the key isn't in the dictionary, you get an exception:

```
>>> print (eng2sp['four'])
KeyError: 'four'
```

5.16 Accessing Elements from Dictionary

While indexing is used with other data types to access values, a dictionary uses keys. Keys can be used either inside square brackets **[]** or with the **get()** method.

If we use the square brackets **[], KeyError** is raised in case a key is not found in the dictionary. On the other hand, the **get()** method returns **None** if the key is not found.

```
# get vs [] for retrieving elements
my_dict = {'name': 'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))

# Trying to access keys which doesn't exist throws error
# Output None
print(my_dict.get('address'))

# KeyError
print(my_dict['address'])
```

Output:

```
Jack
26
None
Traceback (most recent call last):
  File "<string>", line 15, in <module>
    print(my_dict['address'])
KeyError: 'address'
```

5.17 Changing and Adding Dictionary elements

Dictionaries are mutable. We can add new items or change the value of existing items using an assignment operator. If the key is already present, then the existing value gets updated. In case the key is not present, a new **(key: value)** pair is added to the dictionary.

```
# Changing and adding Dictionary Elements
my_dict = {'name': 'Jack', 'age': 26}

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
print(my_dict)
```

Output:

```
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

5.18 Looping Techniques

When looping through dictionaries, the key and corresponding value can be retrieved at the same time using the **items()** method.

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

When looping through a sequence, the position index and corresponding value can be retrieved at the same time using the **enumerate()** function.

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

To loop over two or more sequences at the same time, the entries can be paired with the **zip()** function.

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q, a))
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

5.19 Removing elements from Dictionary

We can remove a particular item in a dictionary by using the **pop()** method. This method removes an item with the provided **key** and returns the **value**.

The **popitem()** method can be used to remove and return an arbitrary **(key, value)** item pair from the dictionary. All the items can be removed at once, using the **clear()** method.

We can also use the **del** keyword to remove individual items or the entire dictionary itself.

```
# Removing elements from a dictionary

# create a dictionary
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# remove a particular item, returns its value
# Output: 16
print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)

# remove an arbitrary item, return (key,value)
# Output: (5, 25)
print(squares.popitem())

# Output: {1: 1, 2: 4, 3: 9}
print(squares)

# remove all items
squares.clear()

# Output: {}
print(squares)

# delete the dictionary itself
del squares

# Throws Error
print(squares)
```

Output

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{}
Traceback (most recent call last):
  File "<string>", line 30, in <module>
    print(squares)
NameError: name 'squares' is not defined
```

5.20 Dictionary Comprehension

Dictionary comprehension is an elegant and concise way to create a new dictionary from an iterable in Python.

Dictionary comprehension consists of an expression pair **(key: value)** followed by a **for** statement inside curly braces **{}**.

Here is an example to make a dictionary with each item being a pair of a number and its square.

```
# Dictionary Comprehension
squares = {x: x*x for x in range(6)}

print(squares)
```

Output

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

5.21 Dictionary Membership Test

We can test if a **key** is in a dictionary or not using the keyword **in**. Notice that the membership test is only for the **keys** and not for the **values**.

```
# Membership Test for Dictionary Keys
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: True
print(1 in squares)

# Output: True
print(2 not in squares)

# membership tests for key only not value
# Output: False
print(49 in squares)
```

Output

```
True
True
False
```

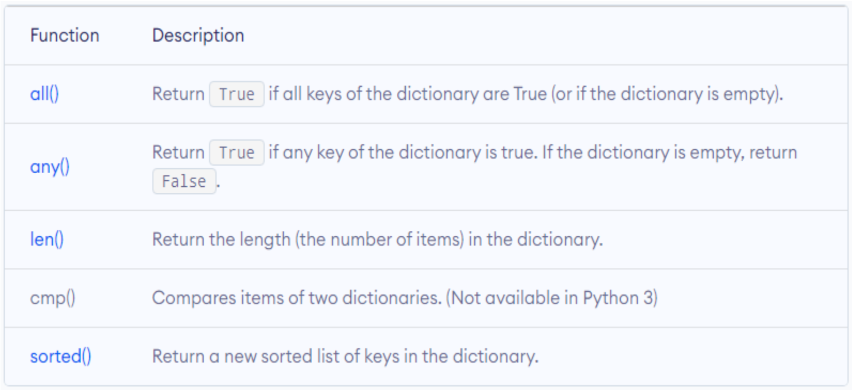
5.22 Dictionary Methods

Methods that are available with a dictionary are tabulated below. Some of them have already been used in the above examples.

Method	Description
clear()	Removes all items from the dictionary.
copy()	Returns a shallow copy of the dictionary.
fromkeys(seq[, v])	Returns a new dictionary with keys from `seq` and value equal to `v` (defaults to `None`).
get(key[,d])	Returns the value of the `key`. If the `key` does not exist, returns `d` (defaults to `None`).
items()	Return a new object of the dictionary's items in (key, value) format.
keys()	Returns a new object of the dictionary's keys.
pop(key[,d])	Removes the item with the `key` and returns its value or `d` if `key` is not found. If `d` is not provided and the `key` is not found, it raises `KeyError`.
popitem()	Removes and returns an arbitrary item (**key, value**). Raises `KeyError` if the dictionary is empty.
setdefault(key[,d])	Returns the corresponding value if the `key` is in the dictionary. If not, inserts the `key` with a value of `d` and returns `d` (defaults to `None`).
update([other])	Updates the dictionary with the key/value pairs from `other`, overwriting existing keys.
values()	Returns a new object of the dictionary's values

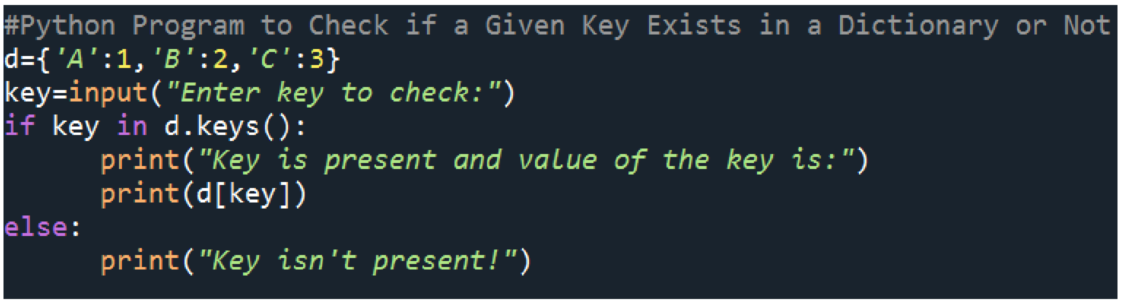
5.23 Dictionary Built-in Functions

Built-in functions like **all(), any(), len(), cmp(), sorted(),** etc. are commonly used with dictionaries to perform different tasks.



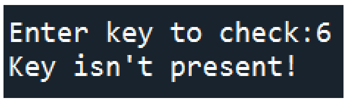
Function	Description
all()	Return `True` if all keys of the dictionary are True (or if the dictionary is empty).
any()	Return `True` if any key of the dictionary is true. If the dictionary is empty, return `False`.
len()	Return the length (the number of items) in the dictionary.
cmp()	Compares items of two dictionaries. (Not available in Python 3)
sorted()	Return a new sorted list of keys in the dictionary.

Example:1



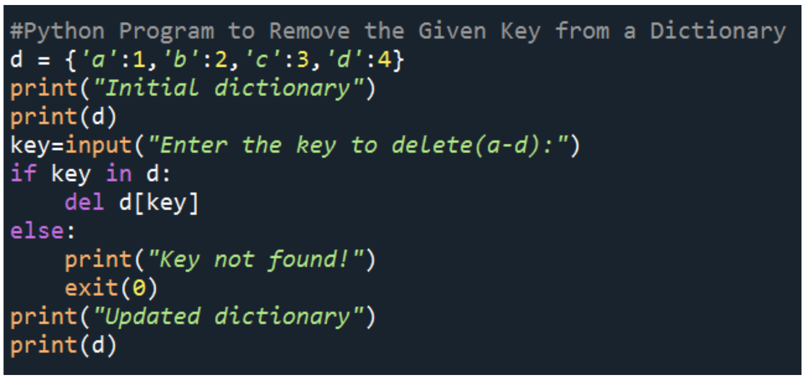
```
#Python Program to Check if a Given Key Exists in a Dictionary or Not
d={'A':1,'B':2,'C':3}
key=input("Enter key to check:")
if key in d.keys():
      print("Key is present and value of the key is:")
      print(d[key])
else:
      print("Key isn't present!")
```

Output:



```
Enter key to check:6
Key isn't present!
```

Example:2



```
#Python Program to Remove the Given Key from a Dictionary
d = {'a':1,'b':2,'c':3,'d':4}
print("Initial dictionary")
print(d)
key=input("Enter the key to delete(a-d):")
if key in d:
    del d[key]
else:
    print("Key not found!")
    exit(0)
print("Updated dictionary")
print(d)
```

Output:

```
Initial dictionary
{'a': 1, 'b': 2, 'c': 3, 'd': 4}

Enter the key to delete(a-d):c
Updated dictionary
{'a': 1, 'b': 2, 'd': 4}
```

Example:3

```
#Python Program to Count the Frequency of Words
#Appearing in a String Using a Dictionary
test_string=input("Enter string:")
l=[]
l=test_string.split()
wordfreq=[l.count(p) for p in l]
print(dict(zip(l,wordfreq)))
```

Output:

```
Enter string:Hi, How are you?
{'Hi,': 1, 'How': 1, 'are': 1, 'you?': 1}
```

Objective Questions:

1. What will be the output of above Python code?

```
str1="6/4"

print("str1")
```

   a. 1
   b. 6/4
   c. 1.5
   d. str1

2. Which of the following will result in an error?

```
str1="python"
```

   a. print(str1[2])
   b. str1[1]="x"
   c. print(str1[0:9])
   d. Both (b) and (c)

3. Which of the following is False?
   a. String is immutable.
   b. capitalize() function in string is used to return a string by converting the whole given string into uppercase.
   c. lower() function in string is used to return a string by converting the whole given string into lowercase.
   d. None of these

4. What will be the output of below Python code?

```
str1="Information"

print(str1[2:8])
```

   a. format
   b. formatio
   c. orma
   d. ormat

5. What will be the output of below Python code?

```
str1="Aplication"

str2=str1.replace('a','A')

print(str2)
```

a. application
b. Application
c. ApplicAtion
d. application

6. What will be the output of below Python code?

```
str1="poWer"

str1.upper()

print(str1)
```

a. POWER
b. Power
c. power
d. power

7. What will the below Python code will return?

```
If str1="save paper,save plants"

str1.find("save")
```

a. It returns the first index position of the first occurance of "save" in the given string str1.
b. It returns the last index position of the last occurrence of "save" in the given string str1.
c. It returns the last index position of the first occurrence of "save" in the given string str1.
d. It returns the first index position of the first occurrence of "save" in the given string str1.

8. What will the below Python code will return?

```
list1=[0,2,5,1]

str1="7"

for i in list1:

        str1=str1+i

print(str1)
```

a. 70251
b. 7
c. 15
d. Error

9. Which of the following will give "Simon" as output?

```
If str1="John,Simon,Aryan"
```

a. print(str1[-7:-12])
b. print(str1[-11:-7])
c. print(str1[-11:-6])
d. print(str1[-7:-11])

10. What will following Python code return?

```
str1="Stack of books"
print(len(str1))
```

a. 13
b. 14
c. 15

d. 16

11. Which of these about a dictionary is false?
    a) The values of a dictionary can be accessed using keys
    b) The keys of a dictionary can be accessed using values
    c) Dictionaries aren't ordered
    d) Dictionaries are mutable
12. Which of the following is not a declaration of the dictionary?
    a) {1: 'A', 2: 'B'}
    b) dict([[1,"A"],[2,"B"]])
    c) {1,"A",2"B"}
    d) { }
13. What will be the output of the following Python code snippet?

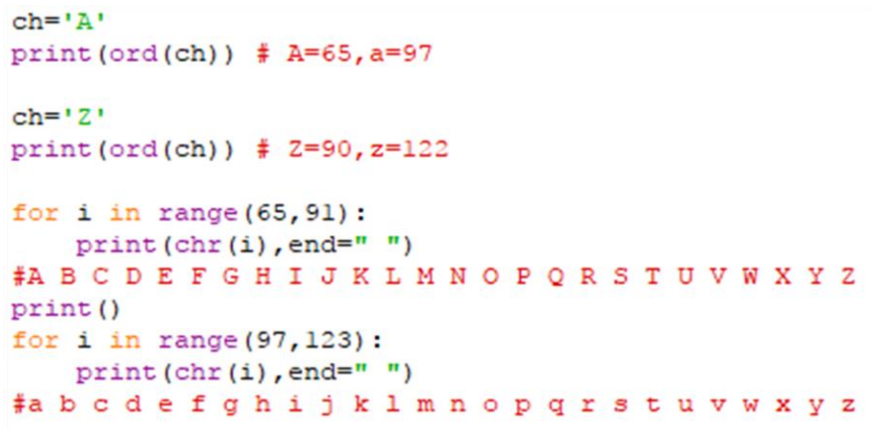
```
a={1:"A",2:"B",3:"C"}
print(a.get(1,4))
```

    a) 1
    b)A
    c)4
    d) Invalid syntax for get method

Solved Problems

1. **Write a program to print Alphabet using ASCII Numbers ?**

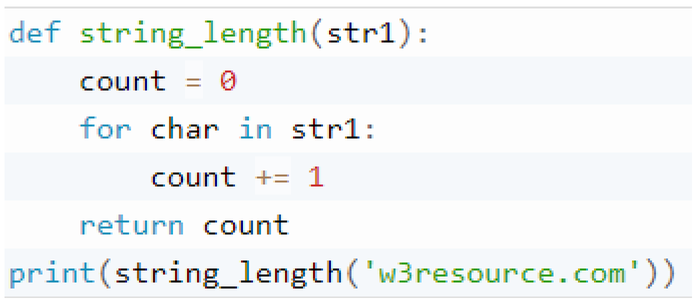
Solutions:-



```
ch='A'
print(ord(ch)) # A=65,a=97

ch='Z'
print(ord(ch)) # Z=90,z=122

for i in range(65,91):
    print(chr(i),end=" ")
#A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
print()
for i in range(97,123):
    print(chr(i),end=" ")
#a b c d e f g h i j k l m n o p q r s t u v w x y z
```

2. **Write a Python program to calculate the length of a string.**

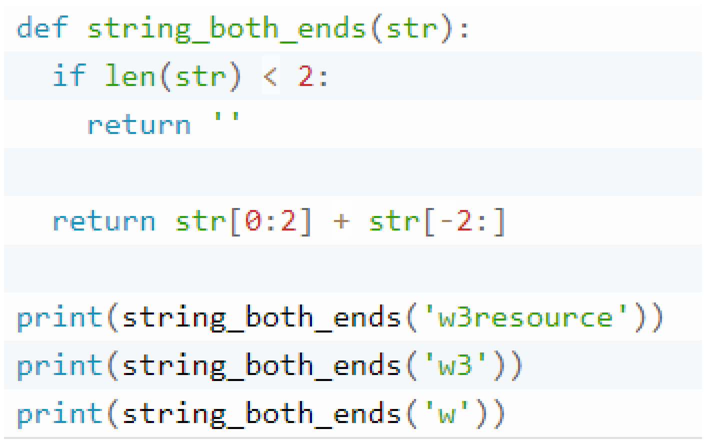
Solutions:-



```
def string_length(str1):
    count = 0
    for char in str1:
        count += 1
    return count
print(string_length('w3resource.com'))
```

3. **Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return instead of the empty string.**

Solutions:-



```python
def string_both_ends(str):
  if len(str) < 2:
    return ''

  return str[0:2] + str[-2:]

print(string_both_ends('w3resource'))
print(string_both_ends('w3'))
print(string_both_ends('w'))
```

Unsolved Problems

1. Write a program to find number of digits ,alphabets and symbols ?
2. Write a program to convert lower case to upper case from given string?
3. Write a program to print the following output?

```
R
R G
R G U
R G U K
R G U K T
R G U K
R G U
R G
R
```

4. Write a program to check whether given string is palindrome or not?
5. Write a program to find no_ words, no_letters, no_digits and no_blanks in a line?
6. Write a program to sort list names in alphabetical order?
7. To find the first character from given string, count the number of times repeated and replaced with * except first character then print final string?
8. To find the strings in a list which are matched with first character equals to last character in a string?
9. Write a program that accepts a string from user and redisplays the same string after removing vowels from it?
10. This is a Python Program to take in two strings and display the larger string without using built-in functions.?
11. Python Program to Read a List of Words and Return the Length of the Longest One?
12. Python Program to Calculate the Number of Upper-Case Letters and Lower-Case Letters in a String?
13. Write a python program to multiply all the items in a dictionary.
14. Python Program to Create a Dictionary with Key as First Character and Value as Words Starting with that Character

UNIT- 6:: FILES Handling