In [1]:
```
# Strings : ordered, immutable, specified with double quotes or single quotes
```

In [2]:
```
# display a string literal with print()

print("MCA HITK")
```

```
MCA HITK
```

In [3]:
```
print('MCA HITK')
```

```
MCA HITK
```

In [4]:
```
print("MCA's are excellent")
```

```
MCA's are excellent
```

In [5]:
```
print("MCA's are \"excellent\"")
```

```
MCA's are "excellent"
```

In [6]:
```
#multiline string
#'''string literal''' or """string literal"""

students = """MCA HITK
Kol"""
print(students)
```

```
MCA HITK
Kol
```

In [7]:
```
#to concatenate, or combine, two strings the + operator can be used

students = "MCA HITK"
place = "Kolkata"
txt = students + "," + " " + place
print(students + ", " + place)
print(students + "," + " " + place)
print(txt)
```

```
MCA HITK, Kolkata
MCA HITK, Kolkata
MCA HITK, Kolkata
```

In [8]:
```
# the * operator can be used to repeat the string for a given number of times

students = "MCA HITK"
place = "Kolkata" * 3
print(students + ", " + place)
```

```
MCA HITK, KolkataKolkataKolkata
```

In [9]:
```
# indexing, length
# square brackets can be used to access elements of the string
# 0  1  2  3  4  5  6  7
# M  C  A     H  I  T  K
#-8 -7 -6 -5 -4 -3 -2 -1

students = "MCA HITK"
print(students[0],students[-6], len(students))
```

```
M A 8
```

In [10]:
```
# iterating through a string

students = "MCA HITK"
for x in students:
    print(x)
```

```
M
C
A

H
I
T
K
```

In [11]:
```
#to check if a certain phrase or character is present in a string, we can use the keyword in
'M' in "MCA"
```

Out[11]:
```
True
```

In [12]:
```
"MC" in "MCA"
```

Out[12]:
```
True
```

In [13]:
```
'B' in "MCA"
```

Out[13]:
```
False
```

In [14]:
```
'MB' in "MCA"
```

Out[14]:
```
False
```

In [15]:
```
#to check if a certain phrase or character is NOT present in a string, use not in
'B' not in "MCA"
```

Out[15]:
```
True
```

In [16]:
```
#slicing strName[startindex : endindex], start index to end index (not included)
# 0  1  2  3  4  5  6  7
# M  C  A     H  I  T  K
#-8 -7 -6 -5 -4 -3 -2 -1

students = "MCA HITK"
print(students[4:8])

#leaving out the start index, will start at the first character

print(students[:7])

#leaving out the end index, will go upto the end

print(students[2:])
```

```
HITK
MCA HIT
A HITK
```

In [17]:
```
students = "MCA HITK"
print(students[:])
```

```
MCA HITK
```

In [18]:
```
#using negative indexes
# M  C  A     H  I  T  K
#-8 -7 -6 -5 -4 -3 -2 -1

students = "MCA HITK"
print(students[-2:])
print(students[-7:-3])
print(students[:-2])
```

```
TK
CA H
MCA HI
```

In [19]:
```
print(dir(str))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute_
_', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__
lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr_
_', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtab
s', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower',
'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'remo
veprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'star
tswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

In [20]:
```
# Python has a set of built-in methods that can be used on strings
# All string methods return new values. They do not change the original string
```

In [20]:
```
help(str.lower)
help(str.upper)
```

```
Help on method_descriptor:

lower(self, /)
    Return a copy of the string converted to lowercase.

Help on method_descriptor:

upper(self, /)
    Return a copy of the string converted to uppercase.
```

In [21]:
```
# lower() method returns the string in lower case
# upper() method returns the string in upper case

students = 'class of MCA'
print(students.lower())
print(students.upper())
```

```
class of mca
CLASS OF MCA
```

In [22]:
```
#'str' object does not support item assignment

students = "MCA HITK"
students[1] = "B"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_20352\3967881319.py in <module>
      2
      3 students = "MCA HITK"
----> 4 students[1] = "B"

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

In [23]:
```
#using the third optional argument 'step' in slicing
# 0  1  2  3  4  5  6  7
# M  C  A     H  I  T  K
#-8 -7 -6 -5 -4 -3 -2 -1

students = "MCA HITK"
print(students[:])
print(students[::2])
print(students[::3])
print(students[::-1])
print(students[::-2])
print(students[::-3])
```

```
MCA HITK
MAHT
M T
KTIH ACM
KI C
KHC
```

In [24]:
```
# reversing a string
import time
students = "MCA HITK"

# Method 1:

length = len(students)
start = time.process_time()
for i in range(100000):
    revStudents = ""
    for x in range(1, length + 1):
        revStudents += students[-x]
end = time.process_time()
print(end - start)
print(revStudents)

# Method 2:
start = time.process_time()
for i in range(100000):
    revStudents = ""
    for x in students:
        revStudents = x + revStudents
end = time.process_time()
print(end - start)
print(revStudents)


# Method 3:
start = time.process_time()
for i in range(100000):
    revStudents = students[::-1]
end = time.process_time()
print(end - start)
print(revStudents)
```

```
0.171875
KTIH ACM
0.125
KTIH ACM
0.015625
KTIH ACM
```

In [25]:
```
help(str.capitalize)
help(str.title)
```

```
Help on method_descriptor:

capitalize(self, /)
    Return a capitalized version of the string.

    More specifically, make the first character have upper case and the rest lower
    case.

Help on method_descriptor:

title(self, /)
    Return a version of the string where each word is titlecased.

    More specifically, words start with uppercased characters and all remaining
    cased characters have lower case.
```

In [26]:
```
# capitalize()
# returns a string where the first character is upper case, and the rest is lower case
```

```
students = "students of MCA 1st year"
print(students.capitalize())

# title()
# returns a string where the first character in every word is upper case
# if the word contains a number or a symbol, the first letter after that will be converted to upper case.

print(students.title())
```

```
Students of mca 1st year
Students Of Mca 1St Year
```

In [27]:
```
help(str.count)
```

```
Help on method_descriptor:

count(...)
    S.count(sub[, start[, end]]) -> int

    Return the number of non-overlapping occurrences of substring sub in
    string S[start:end].  Optional arguments start and end are
    interpreted as in slice notation.
```

In [28]:
```
# count()
# string.count(value, start, end)
# returns the number of times a specified value appears in the string
# the start position and end position are optional, defaults are 0 and end of string respectively

msg = "we forget soon the things we thought we could never forget"
print(msg.count("we"))
print(msg.count("we",10))
print(msg.count("we",10,30))
```

```
3
2
1
```

In [29]:
```
help(str.find)
```

```
Help on method_descriptor:

find(...)
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

In [30]:
```
# find()
# string.find(value, start, end)
# finds the first occurrence (index) of the specified value, returns -1 if the value is not found
# the start position and end position are optional, defaults are 0 and end of string respectively

msg = "we forget soon the things we thought we could never forget"
print('find()')
print(msg.find("we"))
print(msg.find("we",10,30))
print(msg.find("us"))

# index()
# works similar to that of find, but raises an exception if the value is not found

msg = "we forget soon the things we thought we could never forget"
print('index()')
print(msg.index("we"))
print(msg.index("we",10,30))
print(msg.index("us"))
```

```
find()
0
26
-1
index()
0
26
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_20352\1815944259.py in <module>
     17 print(msg.index("we"))
     18 print(msg.index("we",10,30))
---> 19 print(msg.index("us"))

ValueError: substring not found
```

In [31]:
```
help(str.strip)
```

```
Help on method_descriptor:

strip(self, chars=None, /)
    Return a copy of the string with leading and trailing whitespace removed.

    If chars is given and not None, remove characters in chars instead.
```

In [32]:
```
# strip()
# string.strip(characters)
# removes any leading and trailing characters
# the default characters to remove are whitespaces

txt = "      students   "
print("MCA",txt,"love programming")
print("MCA",txt.strip(),"love programming")

txt = "..aa..;@students..."
print("MCA",txt,"love programming")
print("MCA",txt.strip(".;@a"),"love programming")
```

```
MCA       students    love programming
MCA students love programming
MCA ..aa..;@students... love programming
MCA students love programming
```

In [33]:
```
help(str.replace)
```

```
Help on method_descriptor:

replace(self, old, new, count=-1, /)
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.

    If the optional argument count is given, only the first count occurrences are
    replaced.
```

In [34]:
```
# replace()
# string.replace(oldvalue, newvalue, count)
# replaces 'count' occurrences of 'oldvalue' with 'newvalue'
# 'count' is optional. Default is all occurrences

msg = "we forget soon the things we thought we could never forget"
print(msg.replace('we',"I"))
print(msg.replace('we',"I", 1))
print(msg.replace('we',"I", 2))
```

```
I forget soon the things I thought I could never forget
I forget soon the things we thought we could never forget
I forget soon the things I thought we could never forget
```

In [35]:
```
help(str.split)
```

```
Help on method_descriptor:

split(self, /, sep=None, maxsplit=-1)
    Return a list of the words in the string, using sep as the delimiter string.

    sep
      The delimiter according which to split the string.
      None (the default value) means split according to any whitespace,
      and discard empty strings from the result.
    maxsplit
      Maximum number of splits to do.
      -1 (the default value) means no limit.
```

In [36]:
```
# split()
# string.split(separator, maxsplit)
# splits a string into a list
# 'separator' is optional. Default value is any whitespace
# 'maxsplit' is optional. If specified list will contain maxsplit plus one elements
# default value of 'maxsplit' is -1 specifying all occurrences

txt = "MCA HITK Kolkata"
print(txt.split())

txt = "MCA:HITK:Kolkata"
print(txt.split())
print(txt.split(":"))
print(txt.split(":",1))
print("       ".split())
```

```
['MCA', 'HITK', 'Kolkata']
['MCA:HITK:Kolkata']
['MCA', 'HITK', 'Kolkata']
['MCA', 'HITK:Kolkata']
[]
```

In [37]:
```
help(str.join)
```

```
Help on method_descriptor:

join(self, iterable, /)
    Concatenate any number of strings.

    The string whose method is called is inserted in between each given string.
    The result is returned as a new string.

    Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'
```

In [38]:
```python
# join()
# string.join(iterable)
# returns a string by joining all the elements of an iterable,
# a string must be specified as the separator
# iterable : objects capable of returning its members one at a time
# some examples of iterables : String, List, Tuple, Dictionary, Set
# if the iterable contains any non-string values, it raises the TypeError exception

# joining with string
txt = "HITK"
print("".join(txt))
print("-".join(txt))

# joining with list
myList = ['H','I','T','K']
print("".join(myList))
print(":-:".join(myList))
```

```
HITK
H-I-T-K
HITK
H:-:I:-:T:-:K
```

In [39]:
```python
# formatting a string

#formatting with %

print("MCA first year has 58 %s" %'students')

txt = 'students'
print("MCA first year has 58 %s" %txt)

num = 58
print("MCA first year has %d %s" %(num,txt))

age = 21.623
print("MCA first year has %d %s with average age of %f" %(num,txt,age))
print("MCA first year has %d %s with average age of %.2f" %(num,txt,age))
```

```
MCA first year has 58 students
MCA first year has 58 students
MCA first year has 58 students
MCA first year has 58 students with average age of 21.623000
MCA first year has 58 students with average age of 21.62
```

In [40]:
```python
# formatting a string

# using format()
# string.format(value1, value2...)
# The placeholder is defined using curly brackets: {}

print("MCA first year has {} {} with average age of {}".format(num,txt,age))
print("MCA first year has {2} {1} with average age of {0}".format(age,txt,num))
print("MCA first year has {x} {y} with average age of {z}".format(z=age,y=txt,x=num))
print("MCA first year has {} {} with average age of {:.2f}".format(num,txt,age))
```

```
MCA first year has 58 students with average age of 21.623
MCA first year has 58 students with average age of 21.623
MCA first year has 58 students with average age of 21.623
MCA first year has 58 students with average age of 21.62
```

In [41]:
```python
# formatting a string

# using F-strings

print(f"MCA first year has {num} {txt} with average age of {age:.{4}}")
```

```
MCA first year has 58 students with average age of 21.62
```

In [5]:
```python
# Extract the scheme(protocol) from the given url
url = "http://heritageit.edu"
i = url.find(':')
print(url[:i]) #roll:06

n = url.find('.')
print(url[n+1:]) #roll:13

print(url.split(':')[0]) #roll: 56
```

```
http
edu
http
```

In [42]:
```
# Generate a string by rotating a given string by 'n' elements
# Suppose string is "mca students" and n=2
# Left rotation should generate "a studentsmc"
# Right rotation should generate "tsmca studen"
myStr = "mca students"
n = 2
print(myStr[n:] + myStr[:n]) #Roll:11
print(myStr[-n:] + myStr[:-n]) #Roll:23
```

```
a studentsmc
tsmca studen
```

In [43]:
```
# Reverse a string word by word
# Suppose string is "Enjoying Python programming"
# Output should be "programming Python Enjoying"
x = "Enjoying Python programming"
y=x.split()
print(y)
print(" ".join(y[::-1])) #roll:20
```

```
['Enjoying', 'Python', 'programming']
programming Python Enjoying
```

In [ ]: