STRING

- STRING is a sequence of letters or characters

- In python,Strings start and end with single or double quotes.
- >>>”suneel”
- ‘suneel’
- >>>’suneel’
- ‘suneel’

creating string

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

- For example :
- var1 = 'Hello World!'
- var2 = "Python Programming"

Initialize a string variable in Python: “” or None?

- Suppose I have a class with a string instance attribute. Should I initialize this attribute with "" value or None? Is either okay?

  def __init__(self, mystr="")
- self.mystr = mystr

  Or
- def __init__(self, mystr=None)
- self.mystr = mystr

- Edit: What I thought is that if I use "" as an initial value, I "declare" a variable to be of string type. And then I won't be able to assign any other type to it later. Am I right?

- Edit: I think it's important to note here, that my suggestion was WRONG. And there is no problem to assign another type to a variable. I liked a comment of S.Lott: "Since nothing in Python is "declared", you're not thinking about this the right way."

Accessing the elements

- access character in a String:
- >>> s = "python"
- >>> s[3]
- 'h'
- >>> s[6]
- Traceback (most recent call last):
-   File "<stdin>", line 1, in <module>
- IndexError: string index out of range
- >>> s[0]
- 'p'

- s[-1]
- 'n'
- s[-6]
- 'p'
- s[-7]
- Traceback (most recent call last):
- File "<stdin>", line 1, in <module>
- IndexError: string index out of range

Iterating Through String

- Using for loop we can iterate through a string. Here is an example to count the number of 'l' in a string.
- count = 0
- for letter in 'Hello World':
-     if(letter == 'l'):
-         count += 1
- print(count,'letters found')
- 3 letters found

String Membership Test

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

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

Built-in functions to Work with Python

- Various built-in functions that work with sequence, works with string as well.

- Some of the commonly used ones are enumerate() and len(). The enumerate() function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.

- Similarly, len() returns the length (number of characters) of the string.

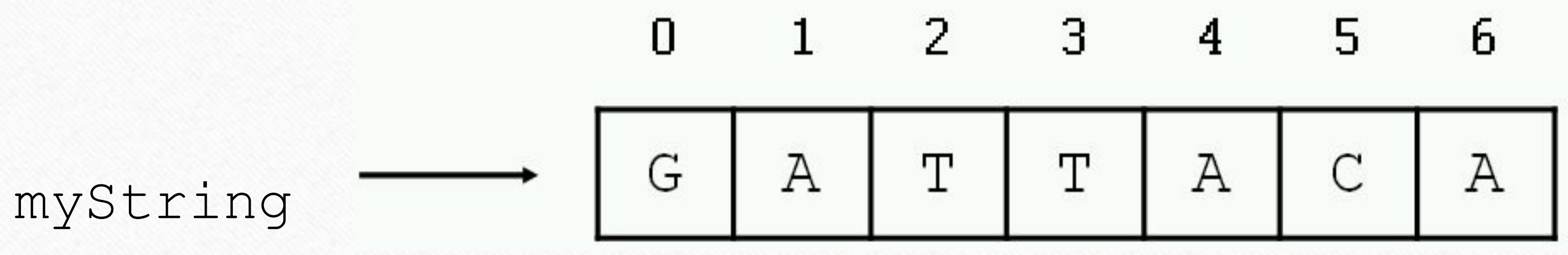
- str = 'cold’

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

  list(enumerate(str) = [(0, 'c'), (1, 'o'), (2, 'l'), (3, 'd')]
- len(str) = 4

Defining strings

- Each string is stored in the computer’s memory as a list of characters.
- `>>> myString = “GATTACA”`



Accessing single characters

- You can access individual characters by using indices in square brackets.

```
myString = "GATTACA"
myString[0]
```

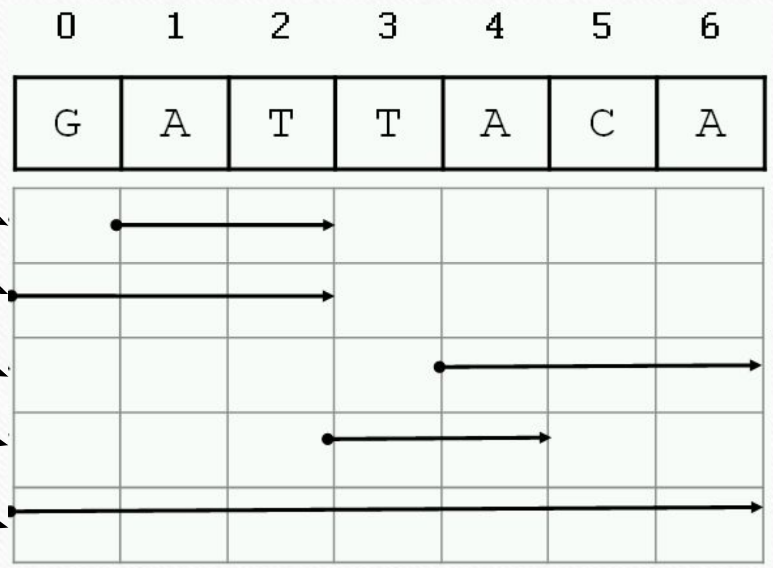
- `'G'`
- `myString[1]`
- `'A'`
- `myString[-1]`
- `'A'`
- `myString[-2]`
- `'C'`
- `myString[7]`

```
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
```

- You can access individual characters by using indices in square brackets

Accessing substrings

- >>> myString = "GATTACA"
- >>> myString[1:3]
- 'AT'
- >>> myString[:3]
- 'GAT'
- >>> myString[4:]
- 'ACA'
- >>> myString[3:5]
- 'TA'
- >>> myString[:]
- 'GATTACA'



String Operator

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

How to create a string in Python?

- Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

- # all of the following are equivalent
- my_string = 'Hello'
- print(my_string)

my_string = "Hello"

- print(my_string)

my_string = '''Hello'''

- print(my_string)

# triple quotes string can extend multiple lines

- my_string = """Hello, welcome to
-             the world of Python"""
- print(my_string)

- Hello
- Hello
- Hello
- Hello, welcome to
-               the world of Python

- In [1]:

STRING METHODS

Learn about all the **string functions** available in **Python** and how you can use them in your program. **String** is a sequence of characters. It's commonly used to represent text. There are number of **methods** defined in **Python** to work with**strings**. ... casefold() - Returns a lowercase **string**, generally used for caseless matching..

LEN

- Python len()   :   Return Length of an Object

- **The len() function returns the number of items (length) in an object.**
- **The syntax of len() is:**
- **len(s)**

len() Parameters

s - a sequence (string, bytes, tuple, list, or range) or a collection (dictionary, set or frozen set)

- **Return Value from len()**

- The len() function returns the number of items of an object.
- Failing to pass an argument or passing an invalid argument will raise a Type error exception

Example 1: How len() works with tuples, lists and range?

testList = []

- print(testList, 'length is', len(testList))
- testList = [1, 2, 3]
- print(testList, 'length is', len(testList))
- testTuple = (1, 2, 3)
- print(testTuple, 'length is', len(testTuple))
- testRange = range(1, 10)
- print('Length of', testRange, 'is', len(testRange))

.

- [ ] length is 0
- [1, 2, 3] length is 3
- (1, 2, 3) length is 3
- Length of range(1, 10) is 9
- In [1]:

CAPITALIZE

- Python String Capitalize : Converts first character to Capital Letter
- **In Python, the capitalize( ) method converts first character of a string to uppercase letter and lowercases all other characters, if any.**
- The syntax of capitalize( ) is:
- string.capitalize( )

capitalize( ) Parameter

The capitalize( ) function doesn't take any parameter.

- **Return Value from capitalize( )**
- The capitalize( ) function returns a string with first letter capitalized and all other characters lowercased. It doesn't modify the original string.

Example 1: Capitalize a Sentence

- string = "python is AWesome.“
- capitalized_string = string.capitalize()
- print('Old String: ', string)
- print('Capitalized String:', capitalized_string)
- Output:
- Old String: python is AWesome.
- Capitalized String: Python is awesome.
- In [1]:

FIND

- Python String find( ) : Returns the index of first occurrence of substring
    - **The find( ) method returns the index of first occurrence of the substring (if found). If not found, it returns -1.**

The syntax of find( ) method is:

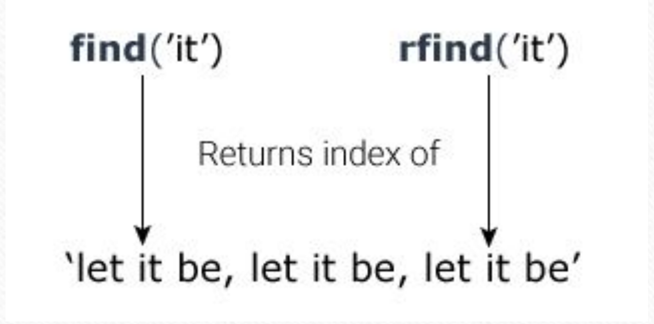
str.find(sub[, start[, end]])

find( ) Parameters

- The find( ) method takes maximum of three parameters:
- **sub** - It's the substring to be searched in the str string
- **start** and **end** (optional) - substring is searched within str[start:end]

Return Value from find( )

- The find( ) method returns an integer value.
- If substring exists inside the string, it returns the index of first occurence of the substring.
- If substring doesn't exist inside the string, it returns -1.



Example 1: find( ) With No start and end Argument

```
quote = 'Let it be, let it be, let it be‘
result = quote.find('let it')
print("Substring 'let it':", result)
result = quote.find('small')
print("Substring 'small ':", result)
# How to use find()
if  (quote.find('be,') != -1):
  print("Contains substring 'be,'")
else:
  print("Doesn't contain substring")
```

- Output:
- Substring 'let it': 11
- Substring 'small ': -1
- Contains substring 'be,'In [1]:

ISALNUM

- Python string isalnum : Checks Alphanumeric Character
- **The isalnum( ) method returns True if all characters in the string are alphanumeric (either alphabets or numbers). If not, it returns False.**
- The syntax of isalnum( ) is:
- string.isalnum( )

isalnum( ) Parameters

The isalnum( ) doesn't take any parameters.

- **Return Value from isalnum( )**
- The isalnum( ) returns:
- **True** if all characters in the string are alphanumeric
- **False** if at least one character is not alphanumeric

Example 1: Working of isalnum( )

- name = "M234onica“
- print(name.isalnum( ))
- # contains whitespace
- name = "M3onica Gell22er “
- print(name.isalnum( ))
- name = "Mo3nicaGell22er“
- print(name.isalnum( ))
- name = "133“
- print(name.isalnum( ))

- Outputs:
- True
- False
- True
- True
- In [1]:

ISALPHA

- Python String isalpha( ) :   Checks if All Characters are Alphabets
- **The isalpha() method returns True if all characters in the string are alphabets. If not, it returns False.**
- The syntax of isalpha( ) is:
- String.isalpha( )
- **isalpha( ) Parameters**
- The isalpha( ) doesn't take any parameters.

- **Return Value from isalpha( )**
- The isalpha( ) returns:
- **True** if all characters in the string are alphabets (can be both lowercase and uppercase).
- **False** if at least one character is not alphabet.

Example 1: Working of isalpha( )

- name = "Monica“
- print(name.isalpha( ))
- # contains whitespace
- name = "Monica Geller“
- print(name.isalpha( ))
- # contains number
- name = "Mo3nicaGell22er“
- print(name.isalpha( ))

- Output:
- True
- False
- False
- In [1]:

ISDIGIT

- Python String isdigit( )   :   Checks Digit Characters
- **The isdigit( ) method returns True if all characters in a string are digits. If not, it returns False.**
- The syntax of isdigit( ) is:
- String.isdigit( )
- **isdigit() Parameters**
- The isdigit( ) doesn't take any parameters.

Return Value from isdigit( )

- The isdigit() returns:
- **True** if all characters in the string are digits.
- **False** if at least one character is not a digit.

Example 1: Working of isdigit( )

- s = "28212“
- print(s.isdigit( ))
- # contains alphabets and spaces
- s = "Mo3 nicaG el l22er“
- print(s.isdigit( ))
- Output:
- True
- False
- In [1]:

LOWER

- Python String lower( ) : returns lowercased string
  - **The string lower( ) method converts all uppercase characters in a string into lowercase characters and returns it.**
  - The syntax of lower( ) method is:
  - String.lower( )
- **String lower( ) Parameters( )**
- The lower( ) method doesn't take any parameters.

Return value from String lower( )

- The lower( ) method returns the lowercased string from the given string. It converts all uppercase characters to lowercase.
- If no uppercase characters exist, it returns the original string.
-

Example 1: Convert a string to lowercase

- # example string
- string = "THIS SHOULD BE LOWERCASE!“
- print(string.lower( ))
- # string with numbers
- # all alphabets whould be lowercase
- string = "Th!s Sh0uLd B3 L0w3rCas3!“
- print(string.lower( ))
- Output:
- this should be lowercase!
- th!s sh0uld b3 l0w3rcas3!

ISLOWER

- Python String islower( )   :   Checks if all Alphabets in a String are Lowercase
- **The islower( ) method returns True if all alphabets in a string are lowercase alphabets. If the string contains at least one uppercase alphabet, it returns False.**
-

  The syntax of islower( ) is:
- String.islower( )

islower( ) parameters

The islower( ) method doesn't take any parameters.

- **Return Value from islower( )**
- The islower( ) method returns:
- True if all alphabets that exist in the string are lowercase alphabets.
- False if the string contains at least one uppercase alphabet.

Example 1: Return Value from islower( )

- s = 'this is good‘
- print(s.islower( ))
- s = 'th!s is a1so g00d‘
- print(s.islower( ))
- s = 'this is Not good‘
- print(s.islower( ))
- Output:
- True
- True
- False
- In [1]:

ISUPPER

- Python String isupper( ) : returns if all characters are uppercase characters
- **The string isupper( ) method returns whether or not all characters in a string are uppercased or not.**
-

  The syntax of isupper( ) method is:
- String.isupper( )
- **String isupper( ) Parameters**
- The isupper( ) method doesn't take any parameters.

Return value from String isupper( )

- The isupper( ) method returns:
- **True** if all characters in a string are uppercase characters
- **False** if any characters in a string are lowercase characters
-

Example 1: Return value of isupper( )

```
# example string
string = "THIS IS GOOD!“
print(string.isupper( ));
# numbers in place of alphabets
string = "THIS IS ALSO G00D!“
print(string.isupper( ));
# lowercase string
string = "THIS IS not GOOD!“
print(string.isupper( ));
```

UPPER

- Python String upper( )   :   returns uppercased string
- **The string upper( ) method converts all lowercase characters in a string into uppercase characters and returns it.**
- The syntax of upper( ) method is:
- String.upper( )
- **String upper( ) Parameters( )**
- The upper( ) method doesn't take any parameters.

Return value from String upper( )

- The upper( ) method returns the uppercased string from the given string. It converts all lowecase characters to uppercase.
- If no lowercase characters exist, it returns the original string.
-

Example 1: Convert a string to uppercase

- # example string
- string = "this should be uppercase!“
- print(string.upper( ))
- # string with numbers
- # all alphabets whould be lowercase
- string = "Th!s Sh0uLd B3 uPp3rCas3!“
- print(string.upper( ))

output

- THIS SHOULD BE UPPERCASE!
- TH!S SH0ULD B3 UPP3RCAS3!
- In [1]:

LSTRIP

- Python Sting istrip( )   :   Removes Leading Characters
- **The lstrip( ) method returns a copy of the string with leading characters removed (based on the string argument passed).**
- The lstrip( ) removes characters from the left based on the argument (a string specifying the set of characters to be removed).

The syntax of lstrip( ) is: string.lstrip([chars])

- **lstrip( ) Parameters**
- chars (optional) - a string specifying the set of characters to be removed.
- If the chars argument is not provided, all leading whitespaces are removed from the string.
- **Return Value from lstrip( )**
- The lstrip( ) returns a copy of the string with leading characters stripped.
- All combinations of characters in the chars argument are removed from the left of the string until first mismatch.

Example: Working of lstrip( )

- random_string = '   this is good ‘
- # Leading whitepsace are removed
- print(random_string.lstrip( ))
- # Argument doesn't contain space
- # No characters are removed.
- print(random_string.lstrip('sti'))
- print(random_string.lstrip('s ti'))
- website = 'https://www.programiz.com/
- 'print(website.lstrip('htps:/.'))

OUTPUT

```
this is good
   this is good
  his is good
   www.programiz.com/
  In [1]:
```

RSTRIP

- Python String rdtrip( )   :   Removes Trailing Characters
- **The rstrip() method returns a copy of the string with trailing characters removed (based on the string argument passed).**
-

  The rstrip() removes characters from the right based on the argument (a string specifying the set of characters to be removed).

The syntax of rstrip() is: string.rstrip([char])

- **rstrip() Parameters**
- chars (optional) - a string specifying the set of characters to be removed.
- If the chars argument is not provided, all whitspaces on the right are removed from the string.
-

Return Value from rstrip()

- The rstrip() returns a copy of the string with trailing characters stripped.
- All combinations of characters in the chars argument are removed from the right of the string until first mismatch.

Example: Working of rstrip()

- random_string = ' this is good‘
- # Leading whitepsace are removed
- print(random_string.rstrip())
- # Argument doesn't contain 'd‘
- # No characters are removed.
- print(random_string.rstrip('si oo'))
- print(random_string.rstrip('sid oo'))
- website = 'www.programiz.com/'print(website.rstrip('m/.'))

OUTPUT

- this is good
- this is good
- this is g
- www.programiz.coIn [1]:

- Output:
- True
- True
- False
- In [1]:

ISSPACE

- Ptthon String isspace( )   :   Checks Whitespace Characters
- **The isspace() method returns True if there are only whitespace characters in the string. If not, it return False.**
- The syntax of isspace() is:
- String.isspace( )
- **isspace() Parameters**
- The isspace() method doesn't take any parameters.

Return Value from isspace()

- The isspace() method returns:
- True if all characters in the string are whitespace characters
- False if the string is empty or contains at least one non-printable() character
-

Example 1: Working of isspace()

- s = '   \t‘
- print(s.isspace())
- s = ' a ‘
- print(s.isspace())
- s = ‘ ‘
- print(s.isspace())

- Output:
- True
- False
- False
- In [1]:

STITLE

- Python String stitle( ) : Checks for Titlecased String
- **The istitle() returns True if the string is a titlecased string. If not, it returns False.**
- **istitle() Parameters**
- The istitle() method doesn't take any parameters.

Return Value from istitle()

- The istitle() method returns:
- True if the string is a titlecased string
- False if the string is not a titlecased string or an empty string

Example 1: Working of istitle()

- s = 'Python Is Good.
- 'print(s.istitle())
- s = 'Python is good'print(s.istitle())
- s = 'This Is @ Symbol.‘
- print(s.istitle())
- s = '99 Is A Number‘
- print(s.istitle())
- s = 'PYTHON‘
- print(s.istitle())

- True
- False
- True
- True
- False
- In [1]:

PARTITION

- **The partition() method splits the string at the first occurrence of the argument string and returns a tuple containing the part the before separator, argument string and the part after the separator.**
- The syntax of partition() is:
- String.partition(saparator)

partition() Parameters()

- The partition() method takes a string parameter separator that separates the string at the first occurrence of it.
- **Return Value from partition()**
- The partition method returns a 3-tuple containing:
- the part before the separator, separator parameter, and the part after the separator if the separator parameter is found in the string
- string itself and two empty strings if the separator parameter is not found

- string = "Python is fun“
- # 'is' separator is found
- print(string.partition('is '))
- # 'not' separator is not found
- print(string.partition('not '))
- string = "Python is fun, isn't it“
- # splits at first occurence of 'is‘
- print(string.partition('is'))

- ('Python ', 'is ', 'fun')
- ('Python is fun', '', '')
- ('Python ', 'is', " fun, isn't it")

REPLACE

- **The replace() method returns a copy of the string where all occurrences of a substring is replaced with another substring.**
- The syntax of replace() is:
- str.replace(old, new[, count])

replace() parameters

- The replace() method can take maximum of 3 parameters:
- **old** - old substring you want to replace
- **new** - new substring which would replace the old substring
- **count** (optional) - the number of times you want to replace the old substring with the new substring
- If count is not specified, replace() method replaces all occurrences of the old substring with the new substring.

Return Value from replace()

- The replace() method returns a copy of the string where old substring is replaced with the new substring. The original string is unchanged.
- If the old substring is not found, it returns the copy of the original string.

- song = 'cold, cold heart‘
- print (song.replace('cold', 'hurt'))
- song = 'Let it be, let it be, let it be, let it be‘
- “'only two occurences of 'let' is replaced"‘
- print(song.replace('let', "don't let", 2))

- hurt, hurt heart
- Let it be, don't let it be, don't let it be, let it be

JOIN( )

- **The join( ) is a string method which returns a string concatenated with the elements of an iterable.**
- The join( ) method provides a flexible way to concatenate string. It concatenates each element of an iterable (such as list, string and tuple) to the string and returns the concatenated string.
- The syntax of join( ) is:
- string.join(iterable)

join( ) Parameters

- The join() method takes an iterable - objects capable of returning its members one at a time
- Some of the example of iterables are:
- **Native datatypes** -List, Tuple, String, Dictionary and set
- File objects and objects you define with an _ iter _ ( ) or getitrem( )_ method

Return Value from join()

- The join() method returns a string concatenated with the elements of an iterable.
- If the iterable contains any non-string values, it raises a TypeError exception.

- numList = ['1', '2', '3', '4']
- seperator = ', ‘
- print(seperator.join(numList))

- numTuple = ('1', '2', '3', '4')
- print(seperator.join(numTuple))
- s1 = 'abc‘
- s2 = '123‘
- """ Each character of s2 is concatenated to the front of s1"""
  print('s1.join(s2):', s1.join(s2))
- """ Each character of s1 is concatenated to the front of s2"""
  print('s2.join(s1):', s2.join(s1))

- 1, 2, 3, 4
- 1, 2, 3, 4
- s1.join(s2): 1abc2abc3
- s2.join(s1): a123b123c

SPLIT( )

- **The split() method breaks up a string at the specified separator and returns a list of strings.**
- The syntax of split() is:
- str.split([separator [, maxsplit]])

split( ) Parameters

- The split() method takes maximum of 2 parameters:
- **separator** (optional)- The is a delimiter. The string splits at the specified separator If the separator is not specified, any whitespace (space, newline etc.) string is a separator.
- **maxsplit** (optional) –The maxsplit defines the maximum number of splits. The default value of maxsplit is -1, meaning, no limit on the number of splits.

Return Value from split()

- The split() breaks the string at the separator and returns a list of strings.
- text= 'Love thy neighbor‘
- # splits at space
- print(text.split())
- grocery = 'Milk, Chicken, Bread‘
- # splits at ',‘
- print(grocery.split(', '))
- # Splitting at ':‘
- print(grocery.split(':'))

Count( )

- **The string count() method returns the number of occurrences of a substring in the given string.**
- In simple words, count() method searches the substring in the given string and returns how many times the substring is present in it.
- It also takes optional parameters start and end to specify the starting and ending positions in the string respectively.
- The syntax of count() method is:
- string.count(substring, start=….., end=……)

String count( ) Parameters

- count() method only requires a single parameter for execution. However, it also has two optional parameters:
- **substring** - string whose count is to be found.
- **start (Optional)** - starting index within the string where search starts.
- **end (Optional)** - ending index within the string where search ends.
- **Note:** Index in Python starts from 0, not 1.

Return value from String count()

count() method returns the number of occurrences of the substring in the given string.

- # define stringstring =
- "Python is awesome, isn't it?“
- substring = "is“
- count = string.count(substring)
- # print count
- print("The count is:", count)
- Output: The count is: 2

ENCODE( )

- **The string encode() method returns encoded version of the given string.**
- Since Python 3.0, string are stored as Unicode, i.e. each character in the string is represented by a code point. So, each string is just a sequence of Unicode code points.
- For efficient storage of these strings, the sequence of code points are converted into set of bytes. The process is known as **encoding**.
- There are various encodings present which treats a string differently. The popular encodings being **utf-8**, **ascii**, etc.
- Using string's encode() method, you can convert unicoded strings into any encoding supported by python By default, Python uses **utf-8** encoding.

By default, encode() method doesn't require any parameters.

The syntax of encode() method is:
string.encode(encoding=‘UTF-8’,errors=‘strict’

- **String encode() Parameters**
- By default, encode() method doesn't require any parameters.
- It returns utf-8 encoded version of the string. In case of failure, it raises a UnicodeDecodeError Exception.

- However, it takes two parameters:
- **encoding** - the encoding type a string has to be encoded to
- **errors** - response when encoding fails. There are six types of error response
  - strict - default response which raises a UnicodeDecodeError exception on failure
  - ignore - ignores the unencodable unicode from the result
  - replace - replaces the unencodable unicode to a question mark **?**
  - xmlcharrefreplace - inserts XML character reference instead of unencodable unicode
  - backslashreplace - inserts a \uNNNN espace sequence instead of unencodable unicode
  - namereplace - inserts a \N{...} escape sequence instead of unencodable unicode
-

Example 1: Encode to Default Utf-8 Encoding

```
# unicode string
 string = 'pythön!‘
 # print string
 print('The string is:', string)
 # default encoding to utf-8
 string_utf = string.encode()
 # print result
 print('The encoded version is:',
string_utf)
```

```
The string is: pythön!
 The encoded version is: b'pyth\xc3\xb6n!'
```

SWAPCASE

- **swapcase( ):**
  - **The string swapcase() method converts all uppercase characters to lowercase and all lowercase characters to uppercase characters of the given string, and returns it.**
  - The format of swapcase( ) method is:
  - String.swapcase( )

String swapcase() Parameters()

The swapcase() method doesn't take any parameters.

- **Return value from String swapcase( )**
- The swapcase( ) method returns the string where all uppercase characters are converted to lowercase, and lowercase characters are converted to uppercase.
-

Example 1: Swap lowercase to uppercase and vice versa using swapcase()

- # example string
- string = "THIS SHOULDALL BE LOWER CASE.
- "print(string.swapcase())
- string = "this should all be uppercase.“
- print(string.swapcase())
- string = "ThIs ShOuLd Be MiXeD cAsEd.“
- print(string.swapcase())

- this should all be lowercase.
- THIS SHOULD ALL BE UPPERCASE.
- tHiS sHoUlD bE mIxEd CaSeD.

STRING CONSTANT

- The constants defined in this module are:
- **ascii_letters**
- The concatenation of the ascii_lowercase and ascii_uppercase constants described below. This value is not locale-dependent.
- **ascii_lowercase**
- The lowercase letters 'abcdefghijklmnopqrstuvwxyz'. This value is not locale-dependent and will not change.
- **ascii_uppercase**
- The uppercase letters 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'. This value is not locale-dependent and will not change.

- **digits**
- The string '0123456789'.
- **hexdigits**
- The string '0123456789abcdefABCDEF'.
- **letters**
- The concatenation of the strings lowercase and uppercase described below. The specific value is locale-dependent, and will be updated when locale.setlocale() is called.
- **lowercase**
- A string containing all the characters that are considered lowercase letters. On most systems this is the string 'abcdefghijklmnopqrstuvwxyz'. Do not change its definition -- the effect on the routines upper() and swapcase() is undefined. The specific value is locale-dependent, and will be updated when locale.setlocale() is called.

- **octdigits**
- The string '01234567'.
- **punctuation**
- String of ASCII characters which are considered punctuation characters in the "C" locale.
- **printable**
- String of characters which are considered printable. This is a combination of digits, letters, punctuation, and whitespace.
- **uppercase**
- A string containing all the characters that are considered uppercase letters. On most systems this is the string 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'. Do not change its definition -- the effect on the routines lower() and swapcase() is undefined. The specific value is locale-dependent, and will be updated when locale.setlocale() is called.
- **whitespace**
- A string containing all characters that are considered whitespace. On most systems this includes the characters space, tab, linefeed, return, formfeed, and vertical tab. Do not change its definition -- the effect on the routines strip() and split() is undefined.

REGULAR EXPRESION

- Python RegEx:
- A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.
- RegEx can be used to check if a string contains the specified search pattern.

RegEx Module

- Python has a built-in package called re, which can be used to work with Regular Expressions.
- Import the re module:
- import re

- RegEx in Python
- When you have imported the re module, you can start using regular expressions:
- Example
- Search the string to see if it starts with "The" and ends with "Spain":

```python
import re

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
if (x):
  print("YES! We have a match!")
else:
  print("No match")
```

```
C:\Users\My Name>python demo_regex.py
YES! We have a match!
```

RegEx Functions

The re module offers a set of functions that allows us to search a string for a match:

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

Finding Patterns of Text with Regular Expressions

The previous phone number–finding program works,
but it uses a lot of code to do something limited:
The isPhoneNumber() function is 17 lines but can find only one pattern of phone numbers
. What about a phone number formatted like 415.555.4242 or (415) 555-4242?
What if the phone number had an extension, like 415-555-4242 x99?
The isPhoneNumber() function would fail to validate them. You could add yet more code
for these additional patterns, but there is an easier way.

Regular expressions, called *regexes* for short, are descriptions for a pattern of text.
For example, a \d in a regex stands for a digit character—that is, any single numeral 0 to 9.
The regex \d\d\d-\d\d\d-\d\d\d\d is used by Python to match the same text
the previous isPhoneNumber() function did: a string of three numbers, a hyphen,
three more numbers, another hyphen, and four numbers.
Any other string would not match the \d\d\d-\d\d\d-\d\d \d\d regex.

But regular expressions can be much more sophisticated.
For example, adding a 3in curly brackets ({3}) after a pattern is like saying,
“Match this pattern three times.”
 So the slightly shorter regex \d{3}-\d{3}-\d{4} also matches
the correct phone number format.

- ['Love', 'thy', 'neighbor']
- ['Milk', 'Chicken', 'Bread']
- ['Milk, Chicken, Bread']