Python Complex Data Types (Collections)

➢ Collections in Python are **containers** that are used to store collections of data, for example, list, dict, set, tuple etc. These are built-in collections. Several modules have been developed that provide additional data structures to store collections of data.

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

Strings

- Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes.

- The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.

- Each character is encoded in the ASCII or Unicode character. So we can say that Python strings are also called the collection of Unicode characters.

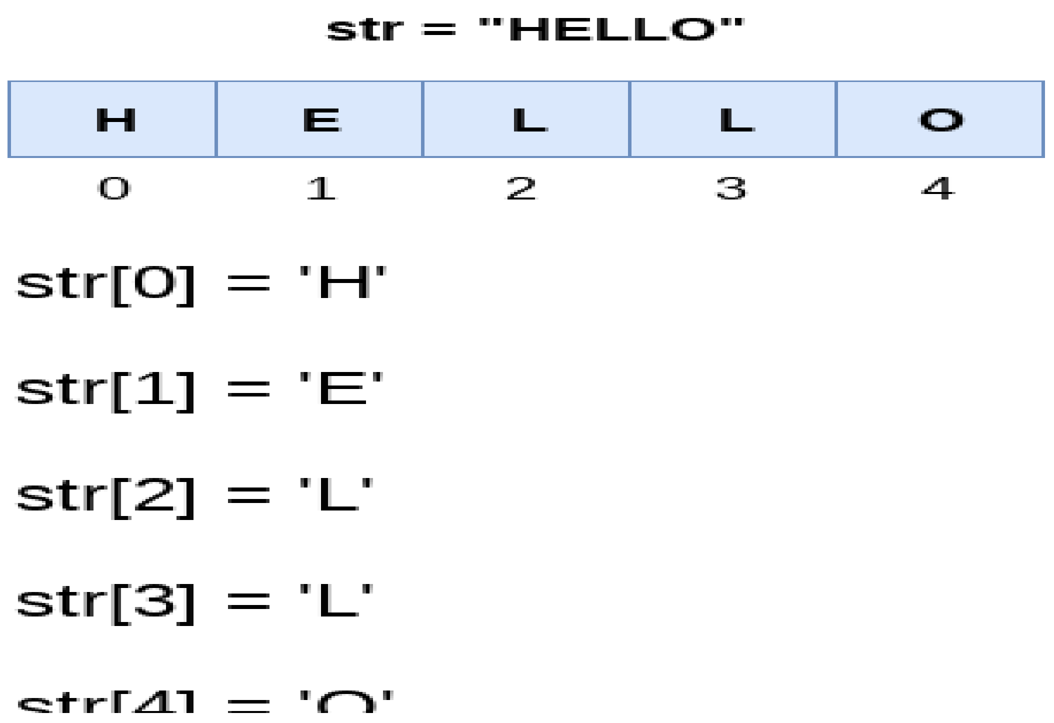
- You can assign a multiline string to a variable by using three Single or Double Quotes.

```
str1 = 'Hello Python'
print(str1)

str2 = "Hello Python"
print(str2)

str3 = ''' Triple quotes are generally used for
           represent the multiline string '''
print(str3)

O/P:  Hello Python
     Hello Python
     Triple quotes are generally used for
      represent the multiline string
```

- Python doesn't support the character data-type; instead, a single character written as 'p' is treated as the string of length 1.

- Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

- Indexing of the Python strings starts from 0. For example, the string "HELLO" is indexed as given below.



```
str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'
```

Square brackets can be used to access elements of the string-

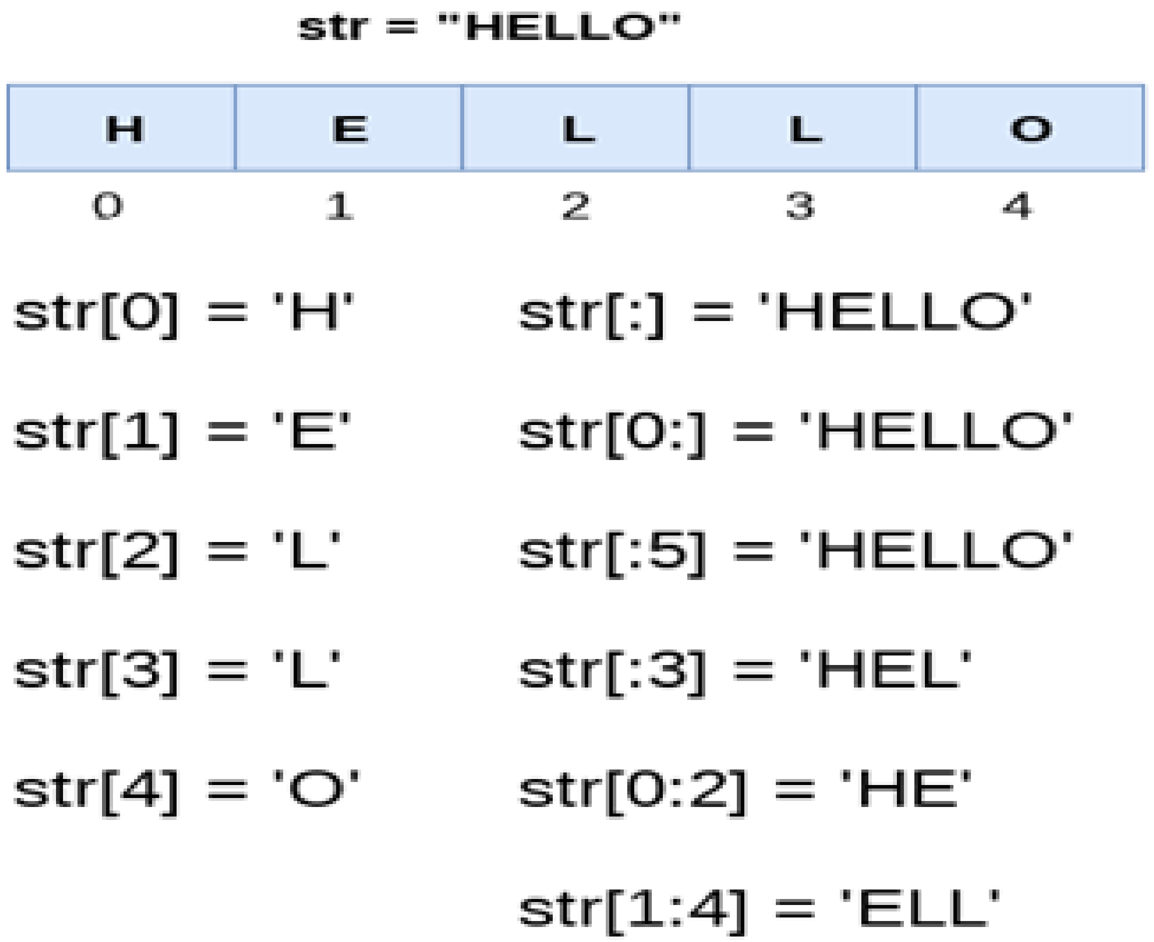
```
str = "HELLO"
print(str[0])   # H
print(str[3])   # L
print(str[6])   # IndexError: string index out of range
```

Slicing:

- You can return a range of characters (Substring) by using the slice syntax.
- Specify the start index and the end index, separated by a colon, to return a part of the string.

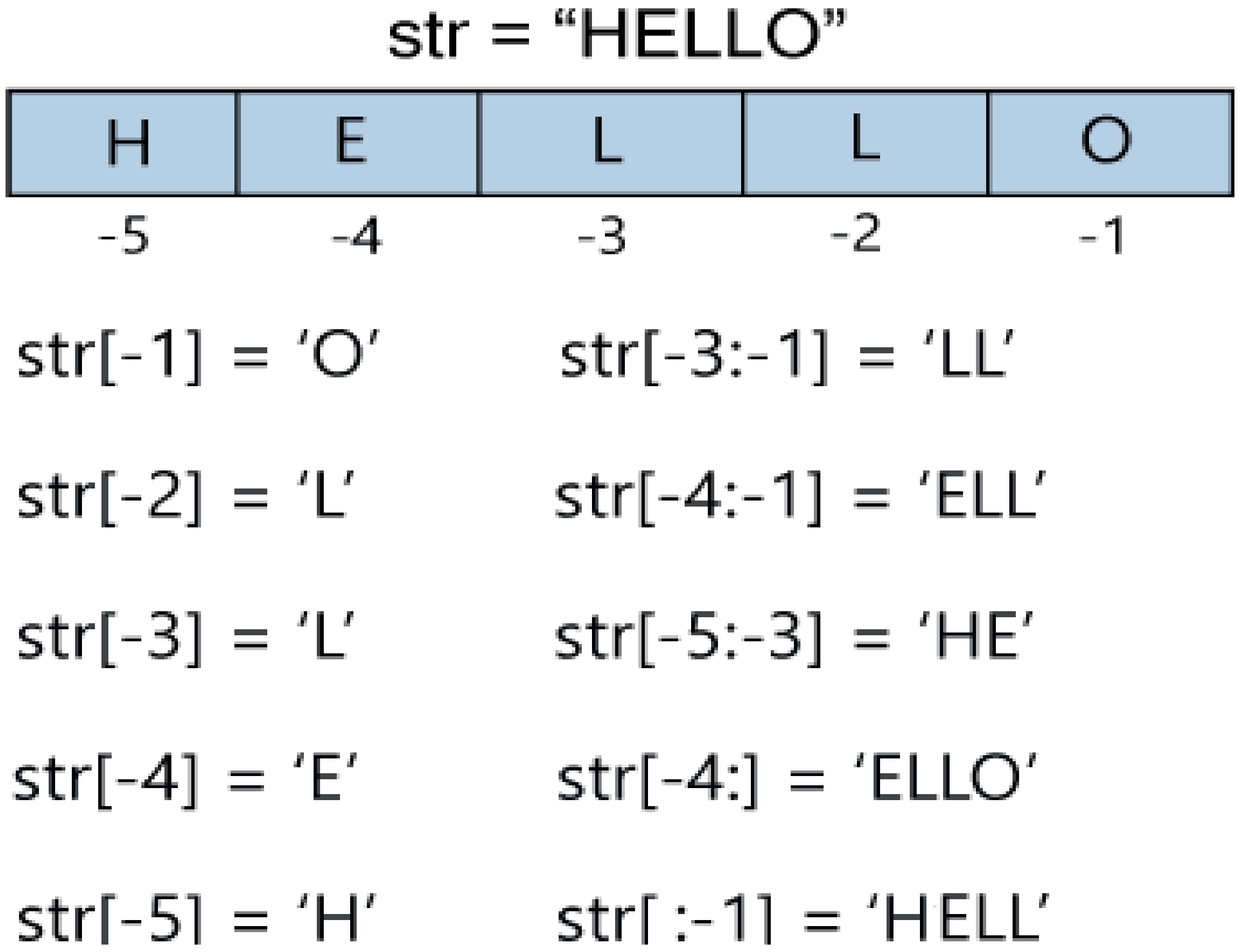
Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])     #llo
```



- Here, we must notice that the upper range given in the slice operator is always exclusive i.e., if str = 'HELLO' is given, then str[1:3] will always include str[1] = 'E', str[2] = 'L' and nothing else.

- We can do the **negative slicing** in the string; it starts from the rightmost character, which is indicated as -1. The second rightmost index indicates -2, and so on.

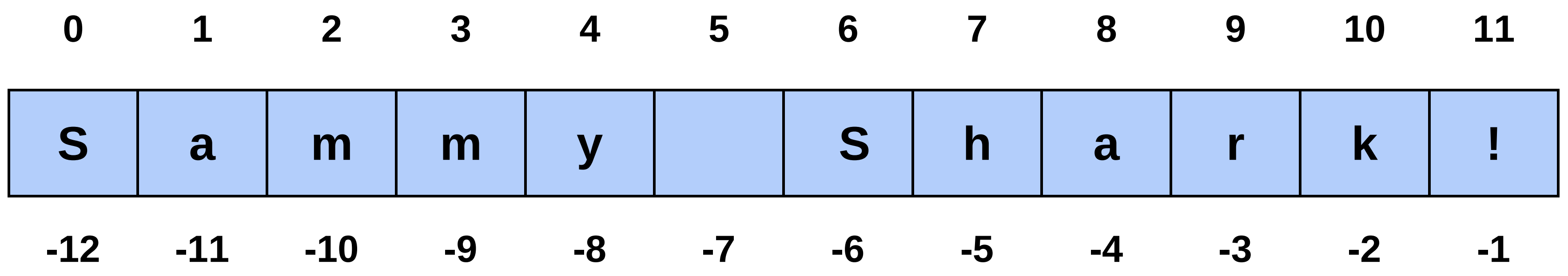


Specifying ‘Stride’ while Slicing Strings:

- String slicing can accept a third (optional) parameter in addition that specifies the **stride**, which refers to how many characters to move forward after the first character is retrieved from the string.

- So far, we have omitted the stride parameter, and Python defaults to the stride of 1, so that every character between two index numbers is retrieved.

s = “Sammy Shark!”



```
s = "Sammy Shark!"
print(s[0:12])          # Sammy Shark!
print(s[0:12:1])      `` # Sammy Shark!
print(s[0:12:2])        # SmySak
print(s[0:12:4])        # Sya
```

- while printing the whole string we can omit the two index numbers and keep the two colons within the syntax to achieve the same result:

  ```
  print(ss[::4])            # Sya
  ```

  Omitting the two index numbers and retaining colons will keep the whole string within range, while adding a final parameter for stride will specify the number of characters to skip

- Additionally, you can indicate a ***negative value for the stride***, which we can use to print the original string in reverse order if we set the stride to `-1`:

  ```
  print(ss[::-1])    # !krahS ymmaS (Reverse Order)
  print(ss[::-2])    # !rh ma (Alternate elements in reverse order)
  ```

```
ss = “Sammy Shark!”
```

0	1	2	3	4	5	6	7	8	9	10	11
S	a	m	m	y		S	h	a	r	k	!
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- In case of negative stride, use right index before colon ’:’ and left index after ‘:’. It is readed from right to left. For example - ss [right : left : -ve]

  (Very important point to understand)

```
print(ss[0:12:-1])       # this will not print anything
print(ss[12:0:-1])       # !krahS ymma
print(ss[6:12:2])        # Sak
print(ss[12:6:-2])       # !rh
print(ss[:-6:-1])        # !krah
```

0	1	2	3	4	5	6	7	8	9	10	11
S	a	m	m	y		S	h	a	r	k	!
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```python
print(ss[-6::-1])        # S ymmaS
print(ss[:-6:-2])        # !rh
print(ss[-6::-2])        # SymS
print(ss[-1:-10:-1])     # !krahS ym
print(ss[-1:-10:1])      # print nothing
print(ss[-4:-10:-1])     # ahS ym
print(ss[-6:-1:1])       # Shark

print(ss[0:12][::-1])    # !krahS ymmaS (nesting)
```

Ex:

```
s="abcdefgh"
print(s[::-1])       # hgfedcba
print(s[::1])        # abcdefgh
print(s[::-2])       # hfdb
print(s[::2])        # aceg
print(s[::-3])       # heb
print(s[::3])        # adg
print(s[0:4][::2])   # ac (nesting)
print(s[0:4][::-2])  # db (nesting)
```

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h
-8	-7	-6	-5	-4	-3	-2	-1

Updating Strings:

- Updation or deletion of characters from a String is not allowed. A string can only be replaced with new string since its content cannot be partially replaced. Strings are immutable in Python.

```
Str=”HELLO”
Str[2]=’A’             # ERROR
```

```
str = "HELLO"
print(str)             #HELLO
str = "hello"
print(str)             #hello
```

Deleting the String:

- As we know that strings are immutable. We cannot delete or remove the characters from the string. But we can delete the entire string using the **del** keyword.

```
str1 = "Python"
del str1
print(str1)   # NameError: name 'str1' is not defined
```

String Operators:

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.

in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values.

Ex:

```
str = "Hello"
str1 = " world"
print(str*3)                          # HelloHelloHello
print(str+str1)                       # Hello world
print(str[4])                         # o
print(str[2:4]);                      # ll
print('w' in str)                     # false (as w is not present in str)
print('wo' not in str1)               # false (as wo is present in str1)
print(r'C://python37')                # C://python37 as it is written
print("The string str : %s"%(str))     # The string str : Hello
```

String Methods:

Method	Description
capitalize()	It capitalizes the first character of the String. This function is deprecated in python3.
casefold()	Converts string into lower case.
center(width , fillchar)	It returns a space padded string with the original string centred with equal number of left and right spaces.
count(Substr, begin, end)	It counts the number of occurrences of a substring in a String between begin and end index.
endswith(suffix , beg, end)	It returns a Boolean value if the string terminates with given suffix between begin and end.
startswith(suffix, beg, end)	It returns a Boolean value if the string starts with given str between begin and end.
expandtabs(tabsize = 8)	It defines tabs in string to multiple spaces. The default space value is 8.
find(Substr , beg, end)	It returns the index value of the string where substring (first

	occurrence) is found between begin index and end index. (returns -1 if the value is not found)
rfind(Substr, beg, end)	It is similar to find but it traverses the string in backward direction.
index(Substr, beg, end)	It throws an exception if string is not found. It works same as find() method.
rindex(Substr, beg, end)	It is same as index but it traverses the string in backward direction.
format(value)	It returns a formatted version of S, using the passed value.
isalnum()	It returns true if the characters in the string are alphanumeric i.e., alphabets or numbers. Otherwise, it returns false. It does not allow special chars even spaces.
isdecimal()	It returns true if all the characters of the string are decimals (0-9).
isdigit()	It returns true if all the characters are digits (0-0), it also return True for some other unicode-supported chars.
isalpha()	It returns true if all the characters are alphabets and there is at

	least one character, otherwise False.
isidentifier()	It returns true if the string is the valid identifier.
islower()	It returns true if the characters of a string are in lower case, otherwise false.
isupper()	It returns true if characters of a string are in Upper case, otherwise False.
isnumeric()	It returns true if the string contains only numeric characters.
isprintable()	It returns true if all the characters of s are printable or s is empty, false otherwise.
isspace()	It returns true if the characters of a string are only white-spaces, otherwise false.
istitle()	It returns true if the string is titled properly and false otherwise. A title string is the one in which the first character of every word is upper-case whereas the other characters are lower-case.
join(seq)	It merges the strings representation of the given sequence.

len(string)	It returns the length of a string.
ljust(width, fillchar)	It returns the space padded strings with the original string left justified to the given width.
rjust(width ,fillchar)	Returns a space padded string having original string right justified to the number of characters specified.
lower()	It converts all the characters of a string to Lower case.
upper()	It converts all the characters of a string to Upper Case.
lstrip(chars)	It removes all leading whitespaces of a string and can also be used to remove particular character from leading.
rstrip(chars)	It removes all trailing whitespace of a string and can also be used to remove particular character from trailing.
strip(chars)	It is used to perform lstrip() and rstrip() on the string.
partition()	It searches for the separator sep in S, and returns the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings.
rpartition()	Same as partition() but it splits the string at the last occurrence

	of seperator substring.
replace(old, new, count)	It replaces the old sequence of characters with the new sequence. The max characters are replaced if max is given.
split(str, maxsplit)	Splits the string according to the delimiter str. The string splits according to the space if the delimiter is not provided. It returns the list of substring concatenated with the delimiter.
rsplit(str, maxsplit)	It is same as split() but it processes the string from the backward direction. It returns the list of words in the string.
splitlines(keeplinebreaks)	It returns the list of strings at each line with newline removed.
swapcase()	It inverts case of all characters in a string.
title()	It is used to convert the string into the title-case i.e., The string **meEruT** will be converted to Meerut.
translate(table)	It translates the string according to the translation table passed in the function .
zfill(width)	Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).

➢ capitalize()

```
txt = "hello, and welcome to my world."
x = txt.capitalize()     # Hello, and welcome to my world.
```

➢ casefold()

```
txt = "Hello, And Welcome To My World!"
x = txt.casefold()       # hello, and welcome to my world!
```

➢ center(length, character)

- length (Required). The length of the returned string
- character (Optional). The character to fill the missing space on each side. Default is space.

```
txt = "abcd"
x = txt.center(12, "O")      # OOOOabcdOOOO
y = txt.center(11, "O")      # OOOOabcdOOO
```

➢ count(value, start, end)

- value (Required). A String. The Substring to search for.
- start (Optional). An Integer. The position to start the search. Default is 0
- end (Optional). An Integer. The position to end the search. Default is the end of the string

```
txt = "I love apples, apple are my favorite fruit"
x = txt.count("apple", 5, 24)   # 2
y = txt.count("apple", 10, 24)  # 1
```

➢ endswith(value, start, end) and startswith(*value*, *start*, *end*)

- value (Required). The value to check if the string ends with
- start (Optional). An Integer specifying at which position to start the search
- end (Optional). An Integer specifying at which position to end the search

```
txt = "Hello, welcome to my world."
print(txt.endswith("my world."))      # True
print(txt.startswith("Hello"))        # True
```

➢ expandtabs(tabsize)

```
txt = "H\te\tl\tl\to"
x = txt.expandtabs()    # H       e       l       l       o
x = txt.expandtabs(2)   # H e l l o
```

➢ find (value, start, end) and rfind (value, start, end)

- value- Required. The value to search for
- start- Optional. Where to start the search. Default is 0
- end- Optional. Where to end the search. Default is end of the string

```
txt = "Hello, welcome to my world, welcome."
print(txt.find("welcome"))      # 7
print(txt.rfind("welcome"))     # 28
```

➢ index(*value, start, end*) and rindex(*value, start, end*)

Python **index()** method is same as the find() method except it returns error on failure. This method returns index of first occurred substring and an error if there is no match found.

- value- Required. The value to search for
- start- Optional. Where to start the search. Default is 0
- end- Optional. Where to end the search. Default is end of the string.

```
txt = "Hello, welcome to my world."
print(txt.index("e", 5, 15))        # 8
print(txt.rindex("e", 5, 15))       # 13
```

➢ format(value1, value2...)

The **format()** method formats the specified value(s) and insert them inside the string's placeholder. The placeholder is defined using curly brackets: {}. The values can be of any data type.

The placeholders can be identified using named indexes {price}, numbered indexes {0}, or even empty placeholders {}.

```
txt1 = "My name is {fname}, I'am {age}".format(fname = "John", age = 36)
txt2 = "My name is {0}, I'am {1}".format("John",36)
txt3 = "My name is {}, I'am {}".format("John",36)
print(txt1)    # My name is John, I'm 36
print(txt2)    # My name is John, I'm 36
print(txt3)    # My name is John, I'm 36
```

➢ Formatting numerical value in different number systems:

```
val = 10
print("decimal: {0:d}".format(val));
# 10 (here 0 is place holder and d represent value in decimal)
print("hex: {0:x}".format(val));        # a
print("octal: {0:o}".format(val));      # 12
print("binary: {0:b}".format(val));     # 1010
```

➢ Formatting float and percentile:

```
val = 100000000
print("decimal: {:,}".format(val));          # decimal: 100,000,000 (formatting float value)
print("decimal: {:%}".format(56/9));         # decimal: 622.222222% formatting percentile value
print("decimal: {:.2%}".format(56/9));       # decimal: 622.22%
print("decimal: {:.3%}".format(56/9));       # decimal: 622.222%
```

➢ isalnum()

```
print("Company  12".isalnum())      #  False
print("Company12".isalnum())        #  True
print("Company_12".isalnum())       #  False
```

➢ isdecimal()

```
print("30".isdecimal())          # True
print("010".isdecimal())         # True
print("47.5".isdecimal())        # False
print("40,000".isdecimal())      # False
```

➢ isdigit()

```
print("30".isdigit())            # True
print("010".isdigit())           # True
print("47.5".isdigit())          # False
print("40,000".isdigit())        # False
```

Main difference between the function **str.isdecimal()** and **str.isdigit()** is that: **str.isdecimal()** return True only for numbers from 0 to 9, at the same time the **str.isdigit()** return True for some other unicode-supported chars.

```
a = "\u0030"        #unicode for 0
b = "\u00B2"        #unicode for ²

print(a.isdecimal())          # True
print(b.isdecimal())          # False

print(a.isdigit())            # True
print(b.isdigit())            # True
```

➢ isalpha()

```
print("Company".isalpha())          # True
print("Company10".isalpha())        # False
```

➢ isidentifier()

```
print("MyFolder".isidentifier())        # True
print("Demo002".isidentifier())         # True
print("2bring".isidentifier())          # False
print("my demo".isidentifier())         # False
```

➢ islower() and isupper()

```
print("abc".islower())             # True
print("ABC".isupper())              # True

print("abcD".islower())            # False
print("abcD".isupper())            # False

print("abc1".islower())            # True
print("ABC1".isupper())            # True
print("1abc".islower())          # True
print("1ABC".isupper())          # True
print("123".islower())           # False
print("123".isupper())           # False
```

➢ isnumeric()

Numeric characters include digit and all the characters which have the Unicode numeric value property. Like ² and ¾ are also considered to be numeric values.

```
print("12345".isnumeric())           # True
print("123abc".isnumeric())          # False
print("123-4525-00".isnumeric())     # False
print("\u0030".isnumeric())          # True (unicode for 0)
print("\u00B2".isnumeric())          # True (unicode for ²)
print("10km2".isnumeric())           # False
```

➢ isprintable()

```
print("Hello, Ptyhon" .isprintable())                            # True
print("Learn Python here\n".isprintable())                       # False
print("\t Python is a programming language".isprintable())       # False
```

➢ isspace()

```
print("    ".isspace())          # True
print("  s  ".isspace())         # False
```

➢ istitle()

```
print("HELLO, AND WELCOME TO MY WORLD".istitle())        # False
print("Hello Abc".istitle())                             # True
print("22 Names".istitle())                              # True
print("This ABC".istitle())                              # False
```

➢ join(*iterable*)

Join all items in a tuple into a string, using a hash character as separator. It allows various iterables like: List, Tuple, String etc.

```
myTuple = ("John", "Peter", "Vicky")
x = "_".join(myTuple)
print(x)                     # John_Peter_Vicky
```

```
list = ['1','2','3']
str=":".join(list)
print(str)                    # 1:2:3

str1="ABCD"
str2="x".join(str1)
print(str2)                   # AxBxCxD
```

- **len(string)**

```
print(len("AB CD"))           # 5
```

➢ ljust(width, fillchar) and rjust(width, fillchar)

- **ljust()** method left justify the string and fill the remaining spaces with fillchars.
- **width**: width of the given string.
- **fillchar**: characters to fill the remaining space in the string. It is **optional**.

```
txt = "Python"
x = txt.ljust(20,'+')
y = txt.rjust(20,'+')
print(x)                # Python++++++++++++++
print(y)                # ++++++++++++++Python
```

➢ lower() and upper()

```
print("Hello my FRIENDS".lower())        # hello my friends
print("Hello my FRIENDS".upper())        # HELLO MY FRIENDS
```

➢ lstrip(chars) and rstrip(*chars*) and strip(*chars*)

- **chars -** optional: A list of chars to remove

```
print("     python     ".lstrip(), "is my favourite")       # python      is my favourite

txt = ",,,,,ssaaww.....python"
x = txt.lstrip(",.asw")
print(x)                                # python

txt = ",,,,,ssaaXww.....python"
x = txt.lstrip(",.asw")
print(x)                                # Xww.....python

txt = "python,,,,,ssqqqww....."
x = txt.rstrip(",.qsw")
print(x)                                # python
```

```
txt = ",,,,,rrttgg.....python....rrr"
x = txt.strip(",.grt")
print(x)                                        # python
```

➢ partition(sep) and rpartition(sep)

It splits the string from the string specified in parameter. It splits the string from at the first occurrence of *parameter* and returns a tuple. The tuple contains the three parts before the separator, the separator itself, and the part after the separator.

It returns an original string and two empty strings, if the seperator not found.

Example:

Search for the word "apple", and return a tuple with three elements:

1 - everything before the "match"

2 - the "match"

3 - everything after the "match"

```
str = "I could eat apple all day"
a = str.partition("apple")
print(a)                              # ('I could eat ', 'apple', ' all day')

txt = "abc xyz def xyz ghi"
x = txt.partition("xyz")
print(x)                              # ('abc ', 'xyz', ' def xyz ghi')

y = txt.rpartition("xyz")
print(y)                              # ('abc xyz def ', 'xyz', ' ghi')
```

```
a = txt.partition("zzz")
print(a)                            # ('abc xyz def xyz ghi', '', '')

b = txt.rpartition("zzz")
print(b)                            # ('', '', 'abc xyz def xyz ghi')
```

➢ replace(old, new, count)

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

```
txt = "I like Java"
x = txt.replace("Java", "Python")
print(x)                         # I like Python
```

```
txt = "one one was a race horse, two two was one too."
x = txt.replace("one", "three", 2)
print(x)                        # three three was a race horse, two two was one too.
```

➢ split(*separator, maxsplit*) and *rsplit(separator, maxsplit)*

- **separator:** Optional. Specifies the separator to use when splitting the string. By default any whitespace is a separator.
- **maxsplit:** Optional. Specifies how many splits to do. Default value is -1, which is "all occurrences".

```
txt = "hello, my name is Peter, I am 26 years old"
x = txt.split(", ")
print(x)                        # ['hello', 'my name is Peter', 'I am 26 years old']
y = txt.rsplit(", ")
print(y)                        # ['hello', 'my name is Peter', 'I am 26 years old']
```

➢ splitlines(*keeplinebreaks*)

- **keeplinebreaks:** Optional. Specifies if the line breaks should be included (True), or not (False). Default value is not (False)

```
txt = "Thank you for the music\nWelcome to the jungle"
x = txt.splitlines()
y = txt.splitlines(True)

print(x)                 # ['Thank you for the music', 'Welcome to the  jungle']
print(y)                 # ['Thank you for the music\n', 'Welcome to the jungle']
print(txt)               # Thank you for the music
                         Welcome to the jungle
```

➢ swapcase()

```
print("Python Program".swapcase())     # pYTHON pROGRAM
```

➢ title()

- It 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("Welcome to my 2nd world".title())# Welcome To My 2Nd World

print("hello b2b2b2 and 3g3g3g".title())   # Hello B2B2B2 And 3G3G3G
```

➢ translate(table)

- It returns a string in which each character has been mapped through the given translation table. We can use maketrans() method to create a translation map from character-to-character mappings in different formats.

```
table = {97 : 103, 101 : None,  111 : 112}
str = "abcdefghijklmnopqrstuvwxyz"
str2 = str.translate(table)
 print(str2)                                  # gbcdfghijklmnppqrstuvwxyz
```

➢ zfill(*len*)

- It adds zeros (0) at the beginning of the string, until it reaches the specified length. It returns original length string if the width is less than string length.

```
a = "hello"
b = "welcome to the jungle"
c = "10.000"
print(a.zfill(10))                  # 00000hello
print(b.zfill(10))                  # welcome to the jungle
print(c.zfill(10))                  # 000010.000
```