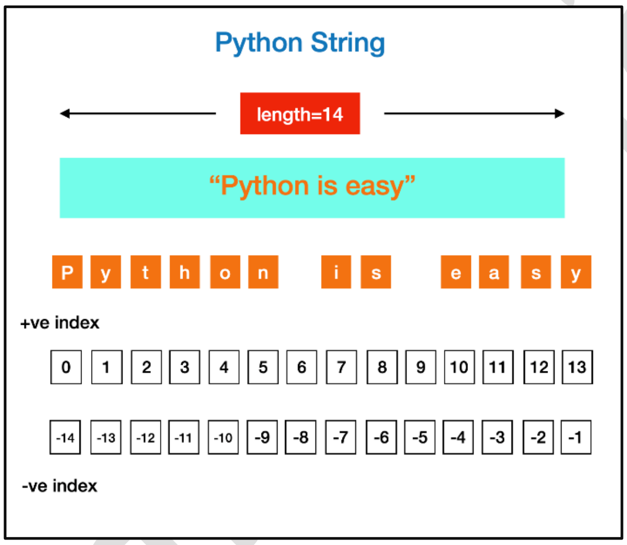
Table of Contents

Chapter 3

Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.
'hello' is the same as "hello".
You can display a string literal with the print() function
Ex.

```
a = "Madi"
a = """Madi
is
perfect """
a = 'Madi'
```

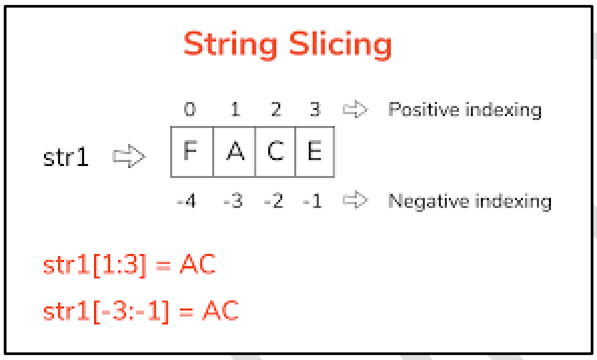


```
# Strings in python are surrounded by either single quotation marks, or double quotation marks.
a = "Madi"
a = """Madi
          is
      perfect """
a = 'Madi'
# Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.
# Get the character at position 1 (remember that the first character has the position 0):
a = "Hello, Madi!"
print(a[1]) #e
# String Length
print("Length of a string: ",len(a)) #12
```

```
# Check in String
# To check if a certain phrase or character is present in a string, we can use the keyword in.
txt = "Madi is perfect"
print("Madi" in txt) #true
print("Madi" not in txt) #false
```

Slicing a string

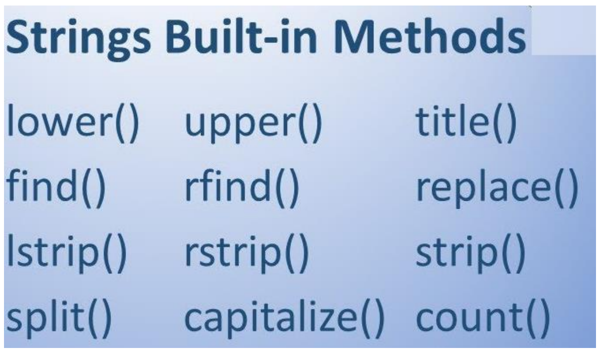
You can return a range of characters by using the slice syntax.
Specify the start index and the end index, separated by a colon, to return a part of the string.



Ex.

```
# Get the characters from position 2 to position 5 (not included):
b = "Hello, Madi!"
print(b[2:5]) #ell
# Get the characters from the start to position 5 (not included):
b = "Hello, Madi!"
print(b[:5]) #Hello
# Get the characters from position 2, and all the way to the end:
b = "Hello, Madi!"
print(b[2:]) #llo, Madi
# Get the characters: From: "a" in "World!" (position -5) To, but not included: "i" in "World!" (position -2):
b = "Hello, Madi!"
#    0123456789..
#    .........-2-1
print(b[-4:-2]) #ad
#    start:not include
b = "Madistic"
print(b[0::2]) # Mdsi
#    start:end:skipAfter
```

String Methods



Strings Built-in Methods

lower()	upper()	title()
find()	rfind()	replace()
lstrip()	rstrip()	strip()
split()	capitalize()	count()

```
# let us create a test string

testString1 = "Hello World!"
print("Original String: "+ testString1)
# Print(this string in lower case)

# Converting a string to lower case
print("Converting to LowerCase")
print(testString1.lower())

# Converting a string to upper case
print("Converting to Upper Case")
print(testString1.upper())

# Capitalizing a string
# Only the first letter in the string will be capitalized
print("Capitalizing the String")
print(testString1.capitalize())

# Trying to slice out a substring between given indexes
print("Substring from index 1 to 7")
print(testString1[1:8])

#Substring from the start till character at index = 7 (start of string is index 0)
print("Substring from the start till character at index = 7 (start of string is index 0):
")
print(testString1[:8])

#Substring from the character at index = 7, till the end of the string (remember:
start of string is index 0)
```

```
print("Substring from the character at index = 7, till the end of the string
(remember: start of string is index 0): ")
print(testString1[7:])


#Find the position of a  substring within the string
#This gives us the first index during a left to right scan. If the string is not found, it
returns -1
print("Find the index from which the substring 'llo' begins within the test string")
print(testString1.find('llo'))

print("Now, let's look for a substring which is not a part of the given string")
print(testString1.find('xxy'))

# Now, trying to find the index of a substring between specified indexes only
print("Now, trying to find a substring between specified indexes only: looking for
'l' between 4 and 9")
print(testString1.find('l',4,9))

# rfind is used, to find the index from the reverse
# So, testString1.rfind('l') will look for the last index of l in the string
print("find('l') on the given string returns the following index (scanning the string
from left to right):")
print(testString1.find('l'))

print("rfind('l') on the given string returns the following index (this scans the string
from right to left):")
print(testString1.rfind('l'))

# Now let us try to replace/substitute a substring of this string with another string
print("Replacing World with Planet")
print(testString1.replace("World","Planet"))

# Now let us try to split the string, into separate words
# let us split it wherever there is a space
print("Splitting the string into words, wherever there is a space")
print(testString1.split(" "))
print(testString1.rsplit(" "))
# Remove leading and trailing whitespace characters
```

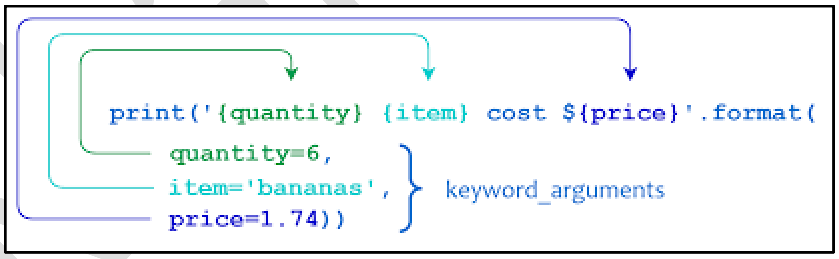
```
testString2 = "Hello World!  "
print("Current Test String=" + testString2)
# print("Length (there are whitespaces at the end):" + len(testString2))
# print("Length after stripping "+ len(testString2.strip()))

# Output of above:
# Original String: Hello World!
# Converting to LowerCase
# hello world!
# Converting to Upper Case
# HELLO WORLD!
# Capitalizing the String
# Hello world!
# Substring from index 1 to 7
# ello Wo
# Substring from the start till character at index = 7 (start of string is index 0):
# Hello Wo
# Substring from the character at index = 7, till the end of the string (remember:
start of string is index 0):
# orld!
# Find the index from which the substring 'llo' begins within the test string
# 2
# Now, let's look for a substring which is not a part of the given string
# -1
# Now, trying to find a substring between specified indexes only: looking for 'l'
between 4 and 9
# -1
# find('l') on the given string returns the following index (scanning the string from
left to right):
# 2
# rfind('l') on the given string returns the following index (this scans the string
from right to left):
# 9
# Replacing World with Planet
# Hello Planet!
# Splitting the string into words, wherever there is a space
# ['Hello', 'World!']
# ['Hello', 'World!']
# Current Test String=Hello World!
```

```
print("Example 2")
# Basic Functions
len('turtle') # 6

# Basic Methods
print('  I am alone '.strip())          # 'I am alone' --> Strips all whitespace
characters from both ends.
print('On an island'.strip('d'))          # 'On an islan' --> # Strips all passed characters
from both ends.
print('but life is good!'.split())         # ['but', 'life', 'is', 'good!']
print('Help me'.replace('me', 'you'))       # 'Help you' --> Replaces first with second
param
print('Need to make fire'.startswith('Need'))# True
print('and cook rice'.endswith('rice'))      # True
print('bye bye'.index('e'))              # 2
print('still there?'.upper())             # STILL THERE?
print('HELLO?!'.lower())                # hello?!
print('ok, I am done.'.capitalize())       # 'Ok, I am done.'
print('oh hi there'.find('i'))            # 4 --> returns the starting index position of the
first occurrence
print('oh hi there'.count('e'))           # 2
```

String Format



```
print('{quantity} {item} cost ${price}'.format(
    quantity=6,
    item='bananas',
    price=1.74))
```

We cannot combine strings and numbers like this:

Ex.

```
age = 18
txt = "My name is Madi, I am " + age
print(txt)
```

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Use the format() method to insert numbers into strings:

Ex.

```
age = 18
```

```
txt = "My name is Madi, and I am {}"
print(txt.format(age))
# My name is Madi, and I am 18
```

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:
Ex.

```
quantity = 3
itemno = 567
price = 499
myorder = "I want to pay {2} rupees for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
# I want to pay 499 rupees for 3 pieces of item 567
```

Python Collections (Arrays)
There are four collection data types in the Python programming language:
-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, unchangeable*, and unindexed. No duplicate members.
-Dictionary is a collection which is ordered** and changeable. No duplicate members.

Strings Methods:

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning