Dr. Labeed Al-Saad

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:

Example:

In [1]:
```
#You can use double or single quotes:

print("Hello")
print('Hello')
```

```
Hello
Hello
```

Quotes Inside Quotes

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example:

In [2]:
```
print("It's alright")
print("He is called 'Johnny'")
print('He is called "Johnny"')
```

```
It's alright
He is called 'Johnny'
He is called "Johnny"
```

Assign String to a Variable

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

Example:

In [3]:
```
a = "Hello"
print(a)
```

```
Hello
```

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example:

You can use three double quotes:

In [7]:
```
a = """This is multiline string,
used three quotes to perform that,
You can write any number of lines between these quotes"""
print(a)
```

```
This is multiline string,
used three quotes to perform that,
You can write any number of lines between these quotes
```

Or three single quotes:

Example:

In [8]:
```
a = '''This is multiline string,
used three quotes to perform that,
You can write any number of lines between these quotes'''
print(a)
```

```
This is multiline string,
used three quotes to perform that,
You can write any number of lines between these quotes
```

**Note: in the result, the line breaks are inserted at the same position as in the code.

Strings are Arrays

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

However, Python does not have a character data type, a single character is simply a string with a length of 1.

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

Example:

Get the character at position 1 **(remember that the first character has the position 0):

In [9]:
```
a = "Hello, World!"
print(a[1])
```

```
e
```

Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a for loop.

Example:

Loop through the letters in the word "banana":

In [10]:
```
for x in "banana":
  print(x)
```

```
b
a
n
a
n
a
```

String Length

To get the length of a string, use the len() function.

Example:

The len() function returns the length of a string:

In [11]:
```
a = "Hello, World!"
print(len(a))
```

```
13
```

Check String

To check if a certain phrase or character is present in a string, we can use the keyword in.

Example:

Check if "free" is present in the following text:

In [12]:
```
txt = "The best things in life are free!"
print("free" in txt)
```

```
True
```

Use it in an if statement:

Example:

Print only if "free" is present:

In [13]:
```
txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")
```

```
Yes, 'free' is present.
```

Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

Example:

Check if "expensive" is NOT present in the following text:

In [14]:
```
txt = "The best things in life are free!"
print("expensive" not in txt)
```

```
True
```

Use it in an if statement:

Example:

print only if "expensive" is NOT present:

In [15]:
```
txt = "The best things in life are free!"
if "expensive" not in txt:
  print("No, 'expensive' is NOT present.")
```

```
No, 'expensive' is NOT present.
```

Python - Slicing Strings

Slicing

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.

**Note: The first character has index 0.

Example:

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

In [16]:
```
b = "Hello, World!"
print(b[2:5])
```

```
llo
```

Slice From the Start

By leaving out the start index, the range will start at the first character:

Example:

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

In [17]:
```
b = "Hello, World!"
print(b[:5])
```

```
Hello
```

Slice To the End

By leaving out the end index, the range will go to the end:

Example:

Get the characters from position 2, and all the way to the end:

In [18]:
```
b = "Hello, World!"
print(b[2:])
```

```
llo, World!
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example:

Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

In [19]:
```
b = "Hello, World!"
print(b[-5:-2])
```

```
orl
```

Python - Modify Strings

Python has a set of built-in methods that you can use on strings.

Upper Case

Example:

The upper() method returns the string in upper case:

In [20]:
```
a = "Hello, World!"
print(a.upper())
```

```
HELLO, WORLD!
```

Lower Case

Example:

The lower() method returns the string in lower case:

In [21]:
```
a = "Hello, World!"
print(a.lower())
```

```
hello, world!
```

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example:

The strip() method removes any whitespace from the beginning or the end:

In [22]:
```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

```
Hello, World!
```

Replace String

Example:

The replace() method replaces a string with another string:

In [24]:
```
a = "Hello, World!"
print(a.replace("l", "J"))
```

```
HeJJo, WorJd!
```

Split String

The split() method returns a list where the text between the specified separator becomes the list items.

Example:

The split() method splits the string into substrings if it finds instances of the separator:

In [25]:
```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

```
['Hello', ' World!']
```

Python - String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Example:

Merge variable a with variable b into variable c:

In [26]:
```
a = "Hello"
b = "World"
```

```
c = a + b
print(c)
```

```
HelloWorld
```

In [ ]:
```
Example:

To add a space between them, add a " ":
```

In [27]:
```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

```
Hello World
```

Python - Format - Strings

String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

Example:

In [28]:
```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[28], line 2
      1 age = 36
----> 2 txt = "My name is John, I am " + age
      3 print(txt)

TypeError: can only concatenate str (not "int") to str
```

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

F-Strings

F-String was introduced in Python 3.6, and is now the preferred way of formatting strings.

To specify a string as an f-string, simply put an f in front of the string literal, and add curly brackets {} as placeholders for variables and other operations.

Example:

Create an f-string:

In [29]:
```
age = 36
txt = f"My name is John, I am {age}"
```

```
print(txt)
```

```
My name is John, I am 36
```

Placeholders and Modifiers

A placeholder can contain variables, operations, functions, and modifiers to format the value.

Example:

Add a placeholder for the price variable:

In [30]:
```
price = 59
txt = f"The price is {price} dollars"
print(txt)
```

```
The price is 59 dollars
```

A placeholder can include a modifier to format the value.

A modifier is included by adding a colon : followed by a legal formatting type, like .2f which means fixed point number with 2 decimals:

Example:

Display the price with 2 decimals:

In [31]:
```
price = 59
txt = f"The price is {price:.2f} dollars"  #.2f which means fixed point number w
print(txt)
```

```
The price is 59.00 dollars
```

A placeholder can contain Python code, like math operations:

Example:

Perform a math operation in the placeholder, and return the result:

In [32]:
```
txt = f"The price is {20 * 59} dollars"
print(txt)
```

```
The price is 1180 dollars
```

Python - Escape Characters

Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

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:

In [33]:
```
txt = "We are the so-called "Vikings" from the north."
```

```
  Cell In[33], line 1
    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:

In [35]:
```
txt = "We are the so-called \"Vikings\" from the north."
print(txt)
```

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

Escape Characters

Other escape characters used in Python:

Code Result ' Single Quote \ Backslash \n New Line \r Carriage Return \t Tab \b Backspace \f Form Feed \ooo Octal value \xhh Hex value

In [36]:
```
txt = 'It\'s alright.'
print(txt)
```

```
It's alright.
```

In [37]:
```
txt = "This will insert one \\ (backslash)."
print(txt)
```

```
This will insert one \ (backslash).
```

In [38]:
```
txt = "Hello\nWorld!"
print(txt)
```

```
Hello
World!
```

In [39]:
```
txt = "Hello\rWorld!"
print(txt)
```

```
World!
```

In [40]:
```
txt = "Hello\tWorld!"
print(txt)
```

```
Hello   World!
```

In [41]:
```
#This example erases one character (backspace):
txt = "Hello \bWorld!"
print(txt)
```

```
Hello World!
```

In [42]:
```
#A backslash followed by three integers will result in a octal value:
txt = "\110\145\154\154\157"
print(txt)
```

```
Hello
```

In [43]:
```
#A backslash followed by an 'x' and a hex number represents a hex value:
txt = "\x48\x65\x6c\x6c\x6f"
print(txt)
```

```
Hello
```

Python - String Methods

String Methods

Python has a set of built-in methods that you can use on strings.

**Note: All string methods return new values. They do not change the original string.

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() Joins the elements of an iterable to the end of the 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.